Parviälykkyys optimointiongelmien ratkaisemisessa

Ilkka Virolainen

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Pro gradu -tutkielma
Ohjaaja: Martti Juhola
Huhtikuu 2015

Tampereen yliopisto
Informaatiotieteiden yksikkö
Tietojenkäsittelyoppi
Ilkka Virolainen: Parviälykkyys optimointiongelmien ratkaisemisessa
Pro gradu -tutkielma, 52 sivua
Huhtikuu 2015

Parviälykkyys optimointiongelmien ratkaisemisessa on saavuttanut viimeisten vuosien aikana alan tutkimuksessa huomattavan aseman. Parviälykkyysalgoritmien toimintaperiaatteet perustuvat luonnossa havaittavaan hyönteisten ja muiden eläinten parvikäyttäytymiseen ja ne tarjoavat useita etuja perinteisiin optimointimenetelmiin nähden. Tässä tutkielmassa esitellään parviälykkyysoptimointia yleisesti ja kuvataan lisäksi tarkemmin joukko valittuja parviälykkyysalgoritmeja.

Avainsanat ja -sanonnat: parviälykkyys, optimointi, metaheuristiikka.

Sisällys

1. Johdanto 1
2. Taustaa 2
3. Optimointi 4
   3.1. NP-kovuus 4
   3.2. Approksimointiheuristiikat 6
   3.3. Metaheuristiikan konsepti ja metaheuristinen optimointi 8
   3.4. Kauppamatkustajan ongelma 10
4. Parviälykkyysalgoritmeja 12
   4.1. Muurahaisyhdyskuntaoptimointi 12
      4.1.1. Metaheuristiikka 15
      4.1.2. Ant System 17
      4.1.3. Variantteja 19
   4.2. Partikkeliparvioptimointi 22
   4.3. Mehiläisyhdyskuntaoptimointialgoritmi 29
   4.4. Harmaasusioptimoija 34
   4.5. Lepakkoalgoritmi 38
   4.6. Käkihaku 42
5. Sovellutuksia ja nykytila 45

Viiteluettelo 48

1. Johdanto

Parviälykkyys tarkoittaa emergenttiä kollektiivista älykkyyttä, joka syntyy yksinkertaisten toimijoiden keskinäisvaikutuksista näiden toimiessa ryhmissä [Bonabeau et al., 1999]. Ilmaisua käytti ensimmäisenä Beni [1988] viitaten solumaisiin robottijärjestelmiin, joissa yksinkertaiset robottitoimijat osoittavat itseorganisoituvaa toimintaa naapurivuorovaikutuksen välityksellä. Sittemmin käsite on omaksuttu käyttöön laajemmassa merkityksessä samalla, kun parviälykkyyttä koskeva tutkimus on yleistynyt. Nyt parviälykkyyttä käytetään yleisesti kuvaamaan hajautettuja ongelmanratkaisumalleja, joiden perustana on luonnossa hyönteisten tai muiden eläinten parvikäyttäytymisessä ilmenevät kollektiiviset mekanismit.

Optimointi on hyvin keskeinen ongelma-alue niin tieteellisessä tutkimuksessa kuin teollisuudessa. Optimoinnilla tarkoitetaan yleisesti parhaan vaihtoehdon valintaa mahdollisten vaihtoehtojen joukosta. Käytännössä ilmeneville optimointiongelmille on tyypillistä, että ne ovat laskennallisesti vaikeita, mistä johtuen tieteellisessä tutkimuksessa esitellään runsaasti erilaisia yrityksiä tuottaa mahdollisimman tehokkaita optimointimenetelmiä. Parviälykkyydestä on viimeisten 20 vuoden aikana muodostunut tärkeä optimointitutkimuksen osa-alue, sillä se on tarjonnut uudenlaisen lähestymistavan vaikeiden ongelmien ratkaisussa: esimääritellyn toiminnallisuuden ja keskitetyn hallinnan sijaan keskiössä ovat emergentti, adaptiivinen hajautettujen toimijoiden käyttäytyminen.

Parviälykkyyden tutkimusalan kasvun myötä on käsitteen määrittelemän kategorian alle luettavien optimointialgoritmien määrä kasvanut valtavasti. Uusissa optimointitekniikoissa poimitaan luonnosta mitä kirjavampia eläimillä ilmeneviä parvikäyttäytymisen mekanismeja ja mallinnetaan niitä matemaattisesti erilaisten optimointiongelmien ratkaisemiseksi. Optimointialgoritmien toimintamekanismit saattavat perustua muurahaisten ravinnonhakuun [Dorigo, 1992], lintujen parveiluun [Kennedy and Eberhart, 1995] tai vaikkapa harmaasusien metsästyskäyttäytymiseen [Mirjalili et al., 2014]. Evoluution luonnonvalinnan kautta hioutuneina toimintamekanismit ovat jo osoittaneet toimintakykynsä käytännössä.

Tämän tutkielman tarkoituksena on tarjota yleiskatsaus parviälykkyyden käytöstä optimointiongelmien ratkaisemisessa. Luvussa 2 esitellään ensin lyhyesti parviälykkyyden taustaa. Luvussa 3 käsitellään optimointia yleisesti, vaikeasti laskettavuuden konseptia ja lähestymistapoja sen ratkaisemiseksi. Luku 4 sisältää tärkeimpien parviälykkyysoptimointialgoritmien käsittelyä, ja luvussa 5 käydään läpi parviälykkyyden tutkimuksen nykytilaa ja sovellusalueita.

2. Taustaa

Parviälykkyyden perustana toimii vahvasti biologian tutkimusala, joka on tutkinut luonnossa ilmenevää eläinparvien itseorganisoituvaa käyttäytymistä. Parvikäyttäytyminen auttaa eläimiä ratkaisemaan luonnossa kohtaamiaan ongelmia, joita ne eivät yksilöinä pystyisi ratkomaan. Lukuisilla lajeilla on parveilukäyttäytymistä muun muassa ravintoa etsiessään ja saalistajia välttäessään. Parvikäyttäytyminen johtaa parempiin selviytymistodennäköisyyksiin kuin mitä yksilöllisen toiminnan kautta olisi mahdollista. Eläimet saavuttavat tavoitteensa kollektiivina käyttäen toimintavälineenään paikallista interaktiota parven tai lauman muiden jäsenten kanssa. Kollektiivinen käyttäytyminen on täysin paikallisten, yksinkertaisten interaktioiden ohjaamaa – se ei ole johtajaorganisoitua tai minkään muun globaalin toimijan sääntelemää. [Zhu et al., 2010]

Historiallisesti hyönteisten parvikäyttäytymisen taustalla vallitsevan mekanismin ajateltiin olevan ensin jumalallista alkuperää, kuten oli asian laita myös monen muun luonnossa esiintyvät monimutkaisen mekanismin suhteen. Darwinin luonnonvalinnan teorian ilmestymisen myötä näille alettiin esittää luonnonvalintaan pohjaavia selityksiä [Beekman et al., 2008]. Nämä selitykset olettivat alun perin toimintamallien mukailevan ihmismäistä käyttäytymistä: hyönteisillä ajateltiin olevan jonkinlainen sisäinen mentaalimalli, jota käyttämällä ne tekevät valintoja tavoitteidensa saavuttamiseksi. Oletettiin, että olisi olemassa suora verrannollisuus asetettujen ongelmien vaikeudessa ja näiden ratkaisemisessa yksilöiltä edellytettävässä kognitiivisten kykyjen tasossa. Tällaiset selitykset ovat sittemmin osoittautuneet hyvin vääriksi: hyönteisyksilöt eivät tarvitse edustamastaan globaalista rakenteesta minkäänlaista mallia. Yksittäisellä hyönteisellä ei ole kykyä hahmottaa tätä ympäröivän tilanteen kokonaiskuvaa, eikä mikään yksilö kontrolloi koko yhdyskunnan toimintaa. Hyönteisyhdyskunta on keskittämätön, itsenäisistä toimijoista koostuva järjestelmä, jonka kokonaistoimintaa voidaan mallintaa yksinkertaisilla todennäköisyysperustaisilla säännöstöillä, joita suoritetaan pelkästään paikallisen informaation perusteella. Yhdyskunnan järjestäytyminen tapahtuu globaalilla tasolla näiden yksinkertaisten sääntöjen sovelluksen seurauksena. Interaktiot johtavat informaation kulkeutumiseen yksilöltä toiselle kulkeutumisen kattaessa koko yhdyskunnan ja tämä informaatio ohjaa samalla yksilön toimintaa yhdyskunnan sisällä. Kokonaisuutena tästä seuraa, että hyönteiset kykenevät yhteiskuntana ratkaisemaan tehokkaasti monenlaisia ongelmia, joiden ratkaisu yksilönä edellyttäisi huomattavia kognitiivisia resursseja. [Garnier et al., 2007]

Ensimmäisiä sovellutuksia hyönteisten parvikäyttäytymismallien käytöstä oli muurahaisyhdyskuntien ravinnonetsintämekanismien soveltaminen optimointiongelmien ratkaisemiseksi [Dorigo, 1992]. Muurahaisyhdyskunnat perustavat ravinnonhakunsa parviälykkyyteen pohjaaviin mekanismeihin, joiden avulla yhdyskunta löytää tehokkaasti pesäänsä lähinnä sijaitsevan ravinnonlähteen. Dorigon mallissa tätä käyttäytymistä hyödynnetään rakentamalla keinotekoisia muurahaisia, jotka etsivät ravinnon kaltaisesti hyviä optimointiongelman ratkaisuja kulkiessaan pitkin abstrakteja graafirakenteita. Sittemmin parviälykkyyteen pohjaten on

kehitetty lukuisia erilaisia menetelmiä, joiden inspiraationa toimivat mitä erilaisimmat luonnon parveilumekanismit.

Parviälyyn pohjaavilla menettelyillä on vaihtoehtoihinsa nähden useita etuja. Ne ovat laskennallisina ongelmanratkaisumenetelminä ensinnäkin usein helppoja ymmärtää ja toteuttaa. Käytetyt mekanismit ovat yksinkertaisia ja perustuvat usein todellisiin, fyysisiin ilmiöihin. Tästä seuraten niitä on verrattain helppo kehittää, mukailla ja yhdistellä muiden tekniikoiden kanssa. Tämän lisäksi ne käsittävät populaatioperustaisina menettelyinä ominaisuuksia, jotka tekevät niistä toimivia ratkaisumalleja optimoinnissa: useista toimijoista koostuvat menettelyt johtavat ratkaisuvaihtoehtojen kattavaan ja tehokkaaseen tutkimiseen. Ne säilövät muistiinsa informaatiota valintavaihtoehdoista ja hyödyntävät sitä tehokkaasti parhaan vaihtoehdon etsimisessä. Parviälykkyysalgoritmeilla on myös pääsääntöisesti vähän kontrolliparametreja, mistä seuraten niiden soveltaminen ongelmatapauksiin on suoraviivaista. [Mirjalili et al., 2014]

Parviälykkyyden tutkimuksen alkuvaiheille tekoälyn ja laskennan kontekstissa Millonas [1993] pyrki määrittelemään ominaispiirteitä, jotka yhdessä määrittelisivät parviälykkyyden parven. Näistä ensimmäinen oli läheisyysperiaate: parven tulisi pystyä suorittamaan tilaan ja aikaan liittyviä laskentaoperaatiota laskentaoperaation tarkoittaessa jotain ympäristön ärsykkeeseen liittyvää reaktiota, joka maksimoi jonkin asian hyödyn parvelle. Toisena periaatteena oli laatuperiaate: parven tulee pystyä vastaamaan myös tehtävän laatuvaatimuksiin. Kolmantena ominaisuutena esitettiin hajautuneisuus, eli parven tulisi hajauttaa resurssejaan eri toimintamalleihin toimintavarmuuden takaamiseksi. Neljäs periaate oli vakausperiaate: parven ei tulisi vaihtaa toimintamalliaan jokaisen ympäristömuutoksen seurauksena energian säästämiseksi. Viimeiseksi esitettiin mukautuvaisuusperiaate: kun ympäristön muutokseen todella kannattaa reagoida, tulisi parven mukautua tähän vastaavasti. Parven on siis tavoiteltava tasapainoa satunnaiskäyttäytymisen ja determinismin välillä. Periaatteita ei ole tarkoitettu tyhjentäväksi listaksi ja sittemmin korkean tason käsitteenä parviälykkyyttä ja sen alaisia menetelmiä on kategorisoitu eri tavoin. Millonaksen esittelemiin periaatteisiin nojataan kuitenkin vielä uusia parviälykkyysalgoritmeja kehitettäessä [Karaboga and Akay, 2009].

3. Optimointi

Optimointi merkitsee parhaan ratkaisun etsimistä kaikkien mahdollisten ratkaisukandidaattien joukosta suhteessa johonkin tavoitefunktioon. Optimointiongelmat ovat erittäin tärkeitä sekä teollisuudessa, että tieteellisessä tutkimuksessa yleensä. Käytännön esimerkkejä optimointiongelmista ovat esimerkiksi kuljetusten, esimerkiksi henkilöjunien, aikataulutus tai tietoverkkojen rakenteen suunnittelu. Muita käytännössä kohdattavia optimointiongelmia ovat esimerkiksi laskennallisessa biologiassa käsiteltävät proteiinimolekyylien laskostumiset [Blum and Li, 2008].

Optimointiongelma P voidaan määritellä kolmikkona

$$P = (S, \Omega, f), \tag{1}$$

missä

1. S on *hakuavaruus*. Hakuavaruus määritellään hyväksikäyttäen äärellistä päätösmuuttujien joukkoa X_i, $i = 1, \dots, n$,
2. Ω on päätösmuuttujia X_i koskevin rajoitteiden joukko, ja
3. $f: S \to \mathbb{R}^+$ on tavoitefunktio, joka antaa jokaiselle ratkaisukandidaatille $s \in S$ vertailtavan arvon.

Mikäli päätösmuuttujien X_i arvoalueet ovat äärellisiä, on optimointiongelma P *kombinatorinen optimointiongelma*. Päätösmuuttujien arvoalueiden ollessa jatkuvat on optimointiongelma *P jatkuva optimointiongelma*. On myös mahdollista, että osalla päätösmuuttujista on jatkuvat ja osalla äärelliset arvoalueet. Tällaisia ongelmia kutsutaan sekamuuttujaongelmiksi (mixed variable optimization). Optimointiongelmassa on tehtävänä etsiä sellainen ratkaisu $s \in S$, että $f(s) \leq f(s'), \forall s' \in S$. Toisin sanoen kaikkien ratkaisukandidaattien joukosta on etsittävä sellainen alkio s, että funktion f arvo tällä syötteellä on pienin mahdollinen. Optimointiongelman voi esittää myös muodossa $f(s) \geq f(s'), \forall s' \in S$, jolloin puhutaan funktion maksimoinnissa. Jokaisen optimointiongelman voi esittää kuitenkin minimointiongelmana, sillä funktion f maksimointi on sama asia kuin funktion $-f$ minimointi [Blum and Roli, 2003]. Optimointiongelmaa kutsutaan *multimodaaliseksi*, mikäli siinä tavoitellaan useamman kuin yhden optimiratkaisun löytämistä [Blum and Li, 2008].

3.1. NP-kovuus

Tärkeä osa tietojenkäsittelytieteiden tutkimusta on tieteenalan alkuvaiheista lähtien ollut laskennallisuuden tutkiminen. Laskennallisuuden tutkimuksessa pyritään selvittämään ongelmien ominaisuuksia ja niiden ratkaisemisen edellytyksiä.

Kun ongelmiin etsitään ratkaisualgoritmeja, ollaan yleensä kiinnostuneita tehokkaimmasta mahdollisesta ratkaisusta. Tehokkuus mitataan yleensä ratkaisun löytämiseen vaaditussa ajassa suhteessa ongelman syötteen kokoon. Algoritmin ajassa mitattava tehokkuus suhteessa ongelman syötteen kokoon tavataan esittää asymptoottisesti rajattuja funktioita hyödyntäen. Tällaisia funktioita kutsutaan *aikakompleksisuusfunktioiksi* ja näiden ilmaisemia asymptoottisia aikakompleksisuuksia käytetään hyväksi muun muassa eri ratkaisualgoritmien välisen tehokkuuden vertailussa ja arvioitaessa, onko jokin tietty ratkaisualgoritmi ylipäätään tehokkuudeltaan sellainen, että ratkaisua voidaan pitää mielekkäänä. Tietyn ratkaisun riittävä tehokkuus on tilannekohtaista, mutta tätä voidaan yleisellä tasolla jaotella sillä, onko ratkaisualgoritmin aikakompleksisuusfunktio polynominen vai ei. Aikakompleksisuus määritellään käyttäen iso-O-, eli Ordo–notaatiota:

$$f(n) = O\big(g(n)\big), \text{ joss. on olemassa sellainen vakio c, että } |f(n)| \leq c|g(n)|, n \geq n_o, \qquad (2)$$

missä $c \in \mathbb{R}^+, n_o \in \mathbb{R}$. Toisin sanoen $f(n)$ on funktion $g(n)$ asymptoottisesti ylhäältä rajaama, jos on olemassa sellainen positiivinen vakio c ja sellainen rajapistevakio n_o, että syötteen kasvaessa ohi rajapisteen funktion $f(n)$ saamat arvot ovat aina pienempiä kuin funktion $g(n)$ saamat arvot. Ratkaisualgoritmin sanotaan olevan *polynominen*, mikäli sen aikakompleksisuusfunktio on $O(p(n))$ jollain polynomisella funktiolla $p(n)$. Mikäli ratkaisualgoritmin aikakompleksisuusfunktio ei ole polynomisesti rajattu, kutsutaan sitä *ylipolynomiseksi*. Jos voidaan osoittaa, että ratkaisu vie vähintään ylipolynomisen ajan, ei tämä ole mielekäs. [Garey and Johnson, 1979]

Vaikeasti laskettavuuden konseptille pohjaavaa *NP-täydellisyyden* teorian tutkimusta pohjusti alun perin Stephen Cook [1971]. Cook kirjoitti *polynomisten palautusten* tärkeydestä laskennallisuuden tutkimuksessa. Algoritmisilla reduktioilla tarkoitetaan menettelyjä, joilla muodostetaan yhden ongelman mielivaltaisesta instanssista toisen ongelman instanssi. Tällainen transformaatio ongelmien välillä mahdollistaa sen, että mikäli transformaation tuloksena olevalle ongelmalle on olemassa ratkaisualgoritmi, voidaan tällä algoritmilla yhdessä ongelman reduktion kanssa ratkaista myös transformaation syötteenä oleva ongelma. Polynomiset reduktiot ovat tällaisia transformaatioita sillä lisäedellytyksellä, että reduktio itsessään on toteutettavissa algoritmilla, joka voidaan suorittaa polynomisesti rajatussa ajassa. Mikäli johonkin ongelmaan on polynomisesti rajatussa ajassa toimiva ratkaisualgoritmi sekä polynomisesti rajatussa ajassa suoritettava reduktio jostain toisesta ongelmasta täksi ongelmaksi, voidaan myös tämä toinen ongelma ratkaista polynomisesti rajatussa ajassa hyödyntäen tätä tunnettua polynomista ratkaisualgoritmia sekä polynomista reduktioita ongelmien välillä.

Laskennallisuuden tutkimuksessa on teknisesti mahdollista ja tarkasteluja yksinkertaistavaa keskittyä päätösongelmien joukkoon NP [Cook, 1971]. Joukkoon NP kuuluvat ongelmat ovat päätösongelmia, jotka voidaan ratkaista polynomisesti rajatussa ajassa epädeterministisellä Turingin koneella. Tähän luokkaan kuuluvat myös useat käytännön tilanteissa kohdattavien ongelmien päätösongelmaversiot. Cook osoitti myös, että jokaiselle ongelmajoukon NP on olemassa

polynominen reduktio *Boolen lausekkeiden toteutuvuusongelmaksi* (Boolean satisfiability problem, SAT). Tästä seuraa, että mikäli Boolen lausekkeiden toteutuvuusongelmalle on olemassa polynomisesti rajatussa ajassa suoritettava ratkaisualgoritmi, voitaisiin tätä algoritmia hyväksikäyttäen ratkaista kaikki joukkoon NP lukeutuvat ongelmat. Sittemmin tätä NP-joukon osajoukon määräävää ekvivalenssiluokkaa on kutsuttu *NP-täydellisten* ongelmien joukoksi [Garey and Johnson, 1979].

Ongelma C on siis NP-täydellinen, mikäli se täyttää kaksi omaisuutta:

1. C kuuluu päätösongelmien joukkoon NP, ja
2. Jokainen NP-täydellinen ongelma voidaan redusoida ongelmaksi C polynomisesti rajatussa ajassa.

Mikäli ongelma C täyttää annetuista ehdoista vain jälkimmäisen, voidaan sitä kutsua *NP-kovaksi* ongelmaksi. NP-kovat ongelmat ovat siis vähintään yhtä vaikeasti ratkaistavissa kuin NP-täydelliset ongelmat. On huomionarvoista, että vaikka laskennallisuuden teorian yhteydessä käsiteltävät ongelmat ovat usein päätösongelmia, on päätösongelmien ja hakuongelmien, joita optimointiongelmat ovat, välillä kiinteä yhteys. Mikäli hakuongelmassa etsitään tietynlaista rakennetta, jolla on jonkinlainen minimikustannus kaikkien mahdollisten rakenteiden joukossa, voidaan vastaava päätösongelma muodostaa käyttäen numeerista raja-arvoa kuvaavaa lisäparametria B, joka lisätään päätösongelman esittämään ongelmanasetteluun kuvaamaan lisävaatimusta: onko olemassa vaaditun kaltainen rakenne, jonka kustannus on korkeintaan B. Maksimointiongelmassa vastaava muunnos toteutuu vastaavasti, mutta parametrin B vertailu käännetään toiseen suuntaan. Tästä seuraa myös, että päätösongelma ei voi olla laskennallisesti vaikeampi kuin vastaava optimointiongelma, olettaen, että ongelmaan liittyvän kustannusfunktion laskeminen on suoritettavissa riittävän tehokkaasti. Tämän päätösongelmien ja optimointiongelmien välillä vallitsevan suhteen myötä NP-täydellisyyden käsitteistöä voidaan soveltaa myös optimointiongelmiin siitä huolimatta, että ongelmaluokka NP ja NP-täydellisyyden määritelmä edellyttää käsiteltävältä ongelmalta sitä, että ongelma on päätösongelma. Tarkemmin voidaan todeta, että mikäli päätösongelma voidaan osoittaa NP-täydelliseksi ja löydetään polynomisesti rajatussa ajassa suoritettava algoritminen reduktio päätösongelmasta toiseksi optimointiongelmaksi, voidaan päätellä, että kyseinen optimointiongelma toteuttaa NP-täydellisen ongelman määrittelyssä esitetyn vaatimuksen 2, ja on siis täten NP-kova. [Garey and Johnson, 1979]

3.2. Approksimointiheuristiikat

Mikäli jokin ongelma todetaan NP-kovaksi tai NP-täydelliseksi seuraa tästä, että on järkevää luopua tavoitteista löytää ongelmalle ratkaisu, joka löytäisi aina optimaalisen ratkaisun polynomisesti rajatussa, eli yleensä käytännön tilanteissa mielekkään lyhyessä, ajassa. Vaihtoehtona on uhrata takuu optimaalisen ratkaisun löytämisestä ja sen sijaan tyytyä etsimään riittävän hyviä ratkaisuja

huomattavasti tehokkaammilla suoritusajoilla. Tämä toteutetaan *heuristisilla algoritmeilla.* Heuristiset algoritmit perustuvat usein järkeviin nyrkkisääntöihin, joita ongelman ratkaisemisessa hyödyntäen on päästy hyväksyttäviin suoritusaikoihin ja ratkaisuihin. Suunnittelumenetelmät heuristisen algoritmien taustalla ovat usein ongelmakohtaisia, joskin on havaittu muutamia yleiskäyttöisiä periaatteita, yhtenä näistä *ympäristöhaku* (neighborhood search), jossa jotain olemassa olevaa ratkaisua parannetaan iteratiivisesti lähentyen jokaisella suoritusiteraatiolla kohti parempia ratkaisuja joidenkin ennalta valittujen operaatioiden mukaisesti, kunnes saavutetaan ratkaisu, jota ei voida enää tällä menettelyllä parantaa. [Garey and Johnson, 1979]

Ympäristöhakuun liittyen voidaan määritellä *naapuruston* käsite hakuavaruudessa. Naapurusto on hakuavaruuden funktio $N: S \rightarrow 2^S$, joka määrittelee jokaisen hakuavaruuden alkiolle naapuruston $N(S) \subseteq S$. Mikäli käsiteltävä optimointiongelma on minimointiongelma, määritellään sen *lokaali optimi* sellaisena ratkaisuna s, että $\forall s' \in N(S): f(s) \leq f(s')$, missä f on optimoitava funktio. Mikäli käsiteltävä optimointiongelma on maksimointiongelma, on sen lokaali optimi vastaavasti sellainen ratkaisu s, että $\forall s' \in N(S): f(s) \geq f(s')$. Lokaalia optimia voidaan kutsua optimointifunktiosta riippuen myös *lokaaliksi minimiksi* tai *lokaaliksi maksimiksi.* Minimointiin tähtäävän optimointiongelman ratkaisua $s \in S$, jolla pätee $f(s) \leq f(s'), \forall s' \in S$, kutsutaan *globaaliksi optimiksi* tai *globaaliksi minimiksi* ja vastaavasti maksimoinnin tapauksessa globaali optimi tai *globaali maksimi* on sellainen ratkaisu $s \in S$, että $f(s) \geq f(s'), \forall s' \in S$ [Dorigo and Stützle, 2004]

Heuristiikka voi perustua ympäristöhaun sijaan myös ratkaisukandidaatin rakentamiseen tyhjästä, jolloin kyseessä on *rakennusalgoritmi.* Rakennusalgoritmit rakentavat ratkaisun optimointiongelmaan askel askeleelta palaamatta rakennusprosessissa taaksepäin. Heuristiikassa aloitetaan tyhjästä ratkaisusta ja lisätään siihen ratkaisukomponentteja, kunnes saavutetaan valmis ratkaisu. Ratkaisukomponentin valinta tehdään jonkin ongelman kannalta sopivan heuristiikan perusteella. Usein tämä heuristiikka on niin kutsuttu *ahne* rakennusheuristiikka, jolloin jokainen komponentti valitaan siten, että tällä saavutetaan annetun heuristisen funktion mukaan valintahetkellä paras arvo. Rakennusalgoritmin periaate on esitetty pseudokoodialgoritmina algoritmissa 1. Algoritmissa proseduuri valitseAhneesti() valitsee ratkaisukomponentin, jolla on jonkin annetun funktion mukaisesti paras heuristinen arvo.

Ratkaisuheuristiikat voidaan jakaa karkeasti kahteen luokkaan sen mukaan, onko näiden taustalla rakennusperiaate vai iteratiivinen ympäristöhaku. Tyypillisiä ongelmia näissä menettelyissä on, että vaikka rakennusalgoritmit ovat laskennallisesti kevyitä, ovat niiden tuottamat ratkaisut usein määrällisesti vähäisiä ja verrattain heikkolaatuisia. Ympäristöhakualgoritmeilla taasen on luontainen taipumus päättää suoritus huonolaatuiseen lokaaliin optimiin. [Dorigo and Stützle, 2004]

Algoritmi AhneRakennusAlgoritmi
Tuloste: Valmis ratkaisu s
    $s_p = ValitseAloitusKomponentti()$
    while s_p *ei täysi ratkaisu* **do**
        $c = ValitseAhneesti(s_p)$
        $s_p = s_p.append(c)$
    end while
    $s = s_p$
    return s
end AhneRakennusAlgoritmi

Algoritmi 1: Ahneen rakennusalgoritmin pseudokoodikuvaus. [Dorigo and Stützle, 2004]

Heurististen algoritmien suorituskyvyn formaali analyysi on näiden luonteesta johtuen vaikeaa. Algoritmit sisältävät usein kokoelman säätöparametreja, jotka ohjaavat algoritmin suoritusta, ja joille etsitään hyvät arvot kokeellisesti. Perinteisillä approksimointialgoritmeilla on hyödynnettävissä todistustekniikoita, joilla voidaan formaalisti todistaa aikakompleksisuustuloksia sekä rajoja, miten lähelle oikeaa ratkaisua algoritmilla päästään. [Garey and Johnson, 1979]

3.3. Metaheuristiikan konsepti ja metaheuristinen optimointi

Metaheuristiikat voidaan määritellä algoritmisena käsitteistönä, jota käytetään hyödyksi muodostettaessa heuristisia menettelyitä laajan ongelmajoukon ratkaisemiseksi. Metaheuristiikkojen voidaan tulkita olevan ylemmän abstraktiotason heuristiikkoja, joita käytetään ohjaamaan heuristiikkojen toimintaa ongelmien ratkaisemiseksi siten, että metaheuristiikalle alisteisten heurististen menetelmien toiminta ohjautuu kohti mahdollisimman hyviä hakuavaruuden alueita. Metaheuristiikkojen kehitys on ollut merkittävänä tekijänä kehitettäessä parempia ratkaisuja vaikeisiin optimointiongelmiin. Näihin lukeutuvat muun muassa esimerkkeinä *tabuhaku* (tabu search), *simuloitu jäähdytys* (simulated annealing) ja *muurahaisyhdyskuntaoptimointi* (ant colony optimization). [Dorigo and Stützle, 2004]

Metaheuristiikan käsitettä on ensimmäisenä käyttänyt Glover [1986] johdanteena löytämistä ja ”jälkeen”-etuliitettä tarkoittavista kreikkalaisperäisistä sanoista heuriskein ja meta. Metaheuristiikoilla voidaan antaa useita toisistaan hieman eriäviä määritelmiä. Blum ja Roli [2003] esittävät metaheuristiikan määrittelevät ominaisuudet seuraavasti:

1. Metaheuristiikat ovat hakuoperaatiota ohjaavia etsintästrategioita.

2. Metaheuristiikkojen tavoitteena on kartoittaa ongelman hakuavaruutta mahdollisimman hyvien ratkaisuiden löytämiseksi.

3. Metaheurististen algoritmien käsittävät toimintamekanismit voivat olla keskenään kovin erilaisia, yksinkertaisia paikallisia hakuja tai monimutkaisempia mekanismeja.

4. Metaheuristiset algoritmit ovat yleensä epädeterministisesti toimivia approksimointialgoritmeja.

5. Metaheuristisiin algoritmeihin sisällytetään usein mekanismeja, joilla vältetään rajoittunut hakuavaruuden läpikäynti.

6. Metaheuristiikat ovat kuvattavissa abstraktilla tasolla.

7. Metaheuristiikat eivät kohdistu yhteen tiettyyn ongelmaan.

8. Metaheuristiikat voivat hyödyntää näille alisteisten heuristisen menetelmien tarjoamaa sovellusaluekohtaista tietoa.

9. Modernit metaheuristiset algoritmit hyödyntävät haussa haun aikana muistiin kerättyä tietoa hakuavaruudesta.

Metaheuristisissa menetelmissä tärkeänä komponenttina pidetään tasapainottelua hakuavaruuden etsimisen ja hakuavaruuden tutkimisen perusteella kerätyn informaation hyödyntämisen kanssa. Näistä tavoitteista puhuttaessa käytetään ilmaisuja *hajauttaminen* (diversification) ja *tehostaminen* (intensification). Vaihtoehtoisesti englanninkielisessä kirjallisuudessa käytetään myös käsitteitä exploration ja exploitation, joskus hieman ensin mainittuja suppeammassa merkityksessä: käsiteltäessä satunnaisperustaisia lyhyen aikavälin strategioita hajauttamisen ja tehostamisen viittaamien pitempien aikavälien strategioiden sijasta. Hajauttamisen yhteydessä pyritään löytämään mahdollisimman tehokkaasti hakuavaruuden osa-alueet, jotka sisältävät hyvälaatuisia ratkaisuja, kun taas tehostamisessa tavoitellaan välttämään jo tutkittuja tai muuten huonoja ratkaisukandidaatteja sisältäviksi havaittuja hakuavaruuden osa-alueita. [Blum and Roli, 2003]

Metaheuristiikkoja voidaan luokitella monella tavalla. Blum ja Roli [2003] ovat esitelleet seuraavan laiset luokittelukategoriat.

1. Luontoon perustuvat / muuhun perustuvat. Metaheuristiikat voidaan luokitella niiden alkuperän perusteella. Luontoon perustuu mm. muurahaisyhdyskuntaoptimoinnin metaheuristiikka, toiseen kategoriaan esimerkiksi tabuhaku.

2. Populaatioperustaiset / pistehakumenetelmät. Metaheuristiikat voidaan luokitella sen mukaan, käsitteleekö se ratkaisun etsimisessä populaatioita vai yksittäistä hakuagenttia.

Hiukkasparvioptimointi edustaa populaatioperustaista menettelyä, pistehakumenetelmiä tai liikeratamenetelmiä edustaa esimerkiksi tabuhaku.

3. Dynaamiset tavoitefunktiot / staattiset tavoitefunktiot. Metaheuristiikat voidaan luokitella tavoitefunktion käyttömenetelmän perusteella. Metaheuristinen menetelmä voi pitää ongelman tavoitefunktion muuttumattomana tai vaihtoehtoisesti muokata tavoitefunktiota sen informaation perusteella, jota hakuavaruudesta on etsintäprosessin myötä saatu.

4. Useita ympäristöjä / yksi ympäristö. Metaheuristiikat voidaan luokitella sen mukaan, käyttävätkö ne ympäristöä tutkiessaan yhtä vai useampaa naapurustoa. Metaheuristiikka voi pitää hakuavaruuden naapurustorakenteen muuttumattomana tai monipuolistaa hakua vaihtamalla naapurustorakennetta haun edetessä.

5. Muistia hyödyntävät / muistia hyödyntämättömät. Metaheuristiikat voidaan luokitella sen mukaan, käyttävätkö ne etsintäprosessissaan avuksi jonkinlaista muistia vai eivät. Ilman muistia toimivat menetelmät hyödyntävät haussa Markovin prosessia, jossa haun seuraavan operaation määrää täysin hakuprosessin senhetkinen tila.

Parviälykkyysalgoritmit ovat määritelmällisesti luontoon perustuvia, muistia hyödyntäviä populaatioperustaisia metaheuristiikkoja. Tavoitefunktioiden tai ympäristörakenteiden staattisuus voi vaihdella heuristiikan mukaan.

3.4. Kauppamatkustajan ongelma

Kauppamatkustajan ongelma (The traveling salesman problem) on yksi tunnetuimmista kombinatoristen optimointiongelmien joukkoon lukeutuvista ongelmista. Sitä käytetään usein optimointialgoritmien suorituskyvyn mittaamiseen. Siinä kauppamatkustajan tehtävänä on etsiä lyhin reitti, joka kulkee jokaisen annetun kaupungin kautta. Tämä ongelma löytyy monien teollisuuden reititysongelmien taustalta. Kauppamatkustajan ongelma on huomionarvoinen kombinatorisia optimointiongelmia käsiteltäessä, sillä se on helposti ymmärrettävissä oleva lyhimmän polun ongelma, jota on tutkittu paljon ja jota käytetään yleisesti ratkaisualgoritmien vertailussa. Näiden lisäksi se lukeutuu NP-kovien ongelmien joukkoon. [Bonabeau et al., 1999]

Kauppamatkustajan ongelma [Blum and Roli, 2003] voidaan määritellä siten, että

1. $G = (V, E)$, missä G on suuntaamaton, täydellinen painotettu graafi, joka koostuu solmujen joukosta V ja kaarien joukosta E.
2. Ongelman tavoitteena on etsiä graafista kaupunkien välisiä etäisyyksiä edustavien kaarien joukosta suljettu polku siten, että jokaisessa solmussa käydään täsmälleen kerran. Tällaista polkua kutsutaan *Hamiltonin sykliksi.*

3. Tavoitteena on etsiä näiden Hamiltonin syklien joukosta sellainen, että polun sisältämin kaarien painotusten summa on pienin mahdollinen.

Tällöin optimointiongelman yleiseen määritykseen pohjautuen hakuavaruus S koostuu kaikista graafin G sisältämistä Hamiltonin sykleistä ja tavoitefunktio $f(s)\colon S \to \mathbb{R}^+$ määritellään ratkaisukomponentin $s \in S$ sisältämien kaarien painojen summana. Usein graafin G kaarien E painotukset ovat symmetriset, siis $c_{ij} = c_{ji}, \forall i,j \in V$, missä c_{ij} on solmujen i ja j välillä kulkevan kaaren paino, mutta voidaan myös määritellä epäsymmetrinen kauppamatkustajan ongelma, jossa välimatkat kaupunkien välillä voivat olla erilaiset suunnasta riippuen [Laporte, 1991].

Kauppamatkustajan ongelma on NP-kova ongelma. Tämä seuraa siitä, että Hamiltonin syklin päätösongelma on NP-täydellinen [Garey and Johnson, 1979] ja on olemassa reduktio Hamiltonin syklin ongelmasta kauppamatkustajan ongelmaksi. Kuten aiemmin on todettu, voidaan NP-täydellisen päätösongelman polynomisesti rajatussa ajassa suoritettavasta reduktiosta optimointiongelmaksi päätellä käsiteltävän optimointiongelman olevan NP-kova. Hamiltonin syklin päätösongelma määritellään siten, että on olemassa graafi $G = (V, E)$ ja kysytään sisältääkö graafi G Hamiltonin syklin. Tästä ongelmasta saadaan reduktio kauppamatkustajan ongelmaan siten, että:

1. Tarkastellaan Hamiltonin syklin päätösongelman tapausta graafille $G^h = (V^h, E^h)$, missä $V^h = \{1, \dots, n\}$ ja $E^h = \{(i,j)\}$.

2. Määritellään kauppamatkustajan ongelman instanssi graafilla $G = (V, E)$, missä $V = V^h$ ja $E = \{(i,j)\colon i,j = 1, \dots, n, i \neq j\}$ ja $c_{ij} = \begin{cases} 1, & (i,j) \in E^h \\ \infty, & (i,j) \notin E^h \end{cases}$.

3. Kohdista 1 ja 2 seuraa, että graafi G sisältää Hamiltonin syklin jos ja vain jos kauppamatkustajan ongelman optimiratkaisu on n [Laporte, 1991].

Kauppamatkustajan ongelman ratkaisemiseksi on kehitetty lukuisia ratkaisualgoritmeja. Osa julkaistuista ratkaisualgoritmeista ovat eksakteja, deterministisiä algoritmeja, jotka tuottavat aina optimimaalisen ratkaisun. Ongelman NP-kovasta luonteesta johtuen on kuitenkin useimmissa tapauksissa mielekkäämpää lähestyä ongelman ratkaisua heuristisen approksimaation kautta, jolloin vältytään usein epäkäytännöllisen pitkiltä suoritusajoilta.

4. Parviälykkyysalgoritmeja

Tässä luvussa on käsiteltyä huomionarvoisimpia parviälykkyysalgoritmeihin lukeutuvia optimointialgoritmeja. Algoritmit eroavat toisistaan monilta ominaisuuksiltaan: ne voivat keskittyä eri ongelma-alueiden ratkaisemiseen, ne pohjaavat erilaisiin mallinnus- ja ongelmanratkaisumenetelmiin ja näiden kaikkien mallinnuspohjana toimii toisistaan erilainen, luonnosta innoituksensa saanut toimintamekanismi. Ne ovat kuitenkin kaikki stokastisia populaatioperustaisia heuristiikkoja optimointiongelmien ratkaisuun. Niiden toiminta perustuu parviälykkyyteen, jossa populaation yksittäisten toimijoiden välisestä interaktiosta syntyy emergentisti yhteiskäyttäytymistä, joka valjastetaan kulloinkin käsiteltävän optimointiongelman ratkaisemiseksi. On huomattava, että vaikka tavoitteena on ollut valita käsittelyyn tärkeimpiä ja muuten huomionarvoisimpia optimointiin kehitettyjä parviälykkyysalgoritmeja, ei listaus ole suinkaan tyhjentävä. Parviälykkyys optimoinnin yhteydessä on viime vuosien aikana kerännyt runsaasti huomiota ja tähän liittyen on kehitetty lukuisia algoritmeja, jotka perustuvat hyvin monenlaisiin luonnossa kohdattaviin toimintamalleihin. Esimerkkeinä käsittelyn ulkopuolelle jääneistä algoritmeista mainittakoon kiiltomatojen parvikäyttäytymiseen perustuva *kiiltomatoparvioptimointi* [Krishnanand and Ghose, 2006], bakteeriyhdyskuntien kultivointiin perustuva *bakteeriparvioptimointi* [Niu and Wang, 2012] ja vesipisaroiden mallinnukseen perustuva *joenmuodostusdynamiikka* [Rabanal et al., 2007]. Pelkästään ampiaisten parvikäyttäytymiseen perustuen on kehitetty lukuisia keskenään kovinkin erilaisia optimointiheuristiikkoja [Karaboga and Akay, 2009].

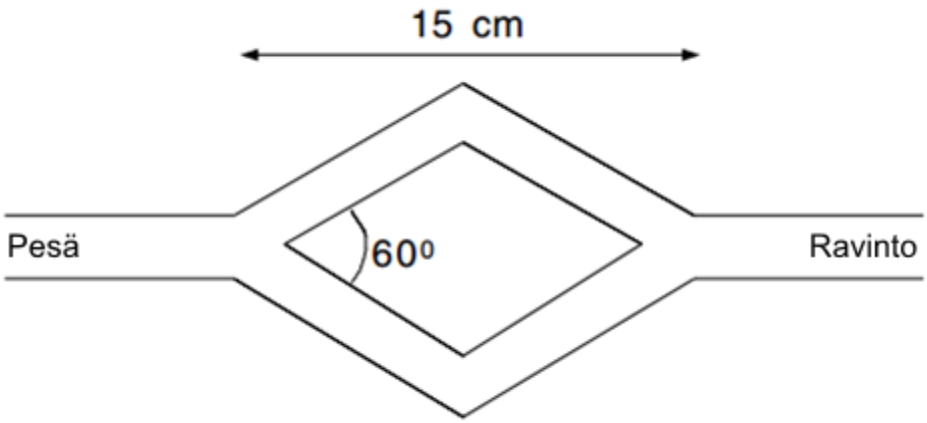
4.1. Muurahaisyhdyskuntaoptimointi

Muurahaisyhdyskuntaoptimointi oli ensimmäisiä ja edelleenkin tunnetuimpia parviälykkyyden sovellutuksia optimointiongelmien ratkaisemiseksi. Alan tutkimuksen pani aluille Marco Dorigo [1992] kollegoineen. Ongelmanratkaisumenettelyn pohjana oli mekanismi, jolla muurahaiset etsivät luonnossa ravintoaan. Erityisen kiinnostava osa muurahaisten ravinnonetsinnässä on ilmiö, jossa muurahaiset löytävät lyhimmän polun pesän ja ravinnonlähteen.

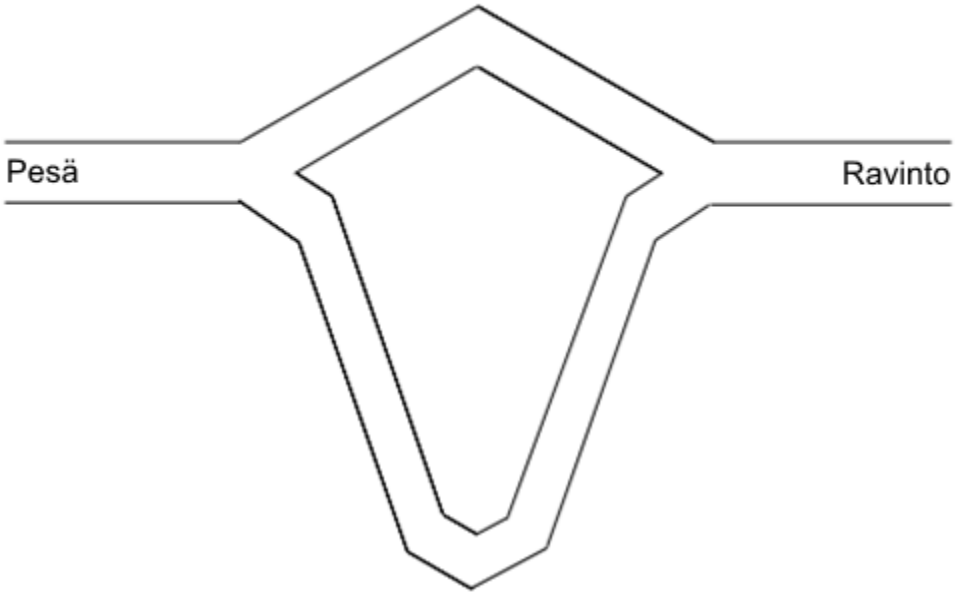
Ranskalainen hyönteistutkija Pierre-Paul Grassé [1946] osoitti monien termiittilajien vastaavaan aiemmin tuntemattomaan ärsykkeeseen. Hän havaitsi myös, että hyönteisten reaktiot tähän ärsykkeeseen voivat tuottaa uusia ärsykkeitä yhdyskunnan muille jäsenille. Grassé kutsui ilmiötä nimellä *stigmergia* ja määritteli, että tämän kommunikaation määrittelevät piirteet ovat epäsuoruus ja paikallisuus: muurahaiset kommunikoivat keskenään muokkaamalla ympäristöään ja muurahaiset, jotka vierailevat ympäristössä, aistivat ärsykkeen ja reagoivat siihen myöhemmällä ajanhetkellä. Tällaista ympäristön muokkaukseen pohjautuvaa epäsuoraa kommunikaatiota muurahaisten välillä on esimerkiksi termiittien pesänrakennus: termiitit rakentavat pesäänsä muodostamalla maa-aineesta ensin liuskoja ja pilareita, sitten kaaria pilareiden väliin ja lopulta pilarien välinen tila täytetään seinien muodostamiseksi. Pesänrakennuksessa ympäristöön tehdyt muokkaukset saavat aikaan ärsykkeitä, jotka saavat yhdyskunnan jäsenet muokkaamaan ympäristöä eri tavoin, jotka taasen

aiheuttavat uusia ärsykkeitä. Toisena, tärkeämpänä esimerkkinä muurahaisten välisestä stigmergisestä kommunikaatiosta toimii feromonijälkiin pohjaava polunetsintämekanismi: yksittäiset muurahaiset erittävät kulkemalleen polulleen feromonia ja muurahaisyhdyskunnan jäsenillä on taipumus valita reitinhaussaan polkuja, joiden feromonipitoisuus on suurempi.

Deneubourg ja muut [1990] osoittivat kokeellisesti muurahaisten ravinnonetsintämekanismin perustuvan stigmergiseen sosiaaliseen itseorganisaatioon ns. *kaksoissiltakokeessa* (double bridge experiment). Kokeessa eroteltiin pesä ruoanlähteestä kahdella yhtä pitkällä sillalla. Koeasetelman alkutilanteessa kummallakaan polulla ei ole feromonijälkiä, mistä johtuen polkua pitkin kulkevilla muurahaisilla on yhtä suuri todennäköisyys valita toinen kahdesta vaihtoehtoisesta polusta. Ajan mittaan toiselle polulle kertyy satunnaisvaihteluista johtuen hieman enemmän muurahaisia, jotka taasen kerryttävät kulkemalleen polulle enemmän feromonia, minkä seurauksena näiden jälkeen tulevat muurahaiset valitsevat todennäköisemmin tämän polun. Koejärjestely on esitelty kuvassa 1 Goss ja muut [1989] varioivat kaksoissiltakoetta siten, että toinen kahdesta polusta on pitempi, samalla osoittaen, että muurahaiset valitsevat useimmiten näistä kahdesta polusta lyhyemmän. Tämä johtuu siitä, että lyhyemmän polun valinneet muurahaiset palaavat ensimmäisenä pesälle, kertyy lyhyemmälle polulle enemmän feromonijälkiä, mikä nostaa seuraavien muurahaisten todennäköisyyttä valita lyhyempi polku. Kaksoissiltakokeen asetelma pidemmällä polulla on havainnollistettu kuvassa 2.



Kuva 1. Kaksoissiltakoe saman pituisilla poluilla [Dorigo and Stützle, 2004]



Kuva 2. Kaksoissiltakoe eripituisilla poluilla [Dorigo and Stützle, 2004]

Denebourg kehitti kollegoidensa kanssa [Deneubourg et al., 1990; Goss et al., 1989] todennäköisyysperustaisen mallin kuvaamaan muurahaisen reitinvalintaa. Mallissa oletetaan, että polulla kulkevien muurahaisten määrä on suoraan verrannollinen polulle kertyneen feromonin määrään, mikä tarkoittaa sitä, ettei feromonin haihtumista oteta huomioon. Muurahaisen todennäköisyys valita polku riippuu suoraan siitä, kuinka monta muurahaista on kulkenut polulla ennen tätä. Malli voidaan esittää kaavoina

$$P_A = \frac{(k + A_i)^n}{(k + A_i)^n + (k + B_i)^n} \tag{3}$$

ja

$$P_A = 1 - P_B, \tag{4}$$

missä P_A ja P_B ovat vastaavasti todennäköisyydet, joilla muurahainen $(i + 1)$ valitsee polun A tai B, A_i ja B_i ovat vastaavasti poluilla A ja B kulkeneiden muurahaisten määrät ja k sekä n ovat säätöparametreja: parametri k määrää merkitsemättömän polun merkityksen siten, että suuremmilla k:n arvoilla polut tarvitsevat suuremmat määrät feromonia, jotta polunvalinta ei olisi enää satunnainen. Säätöparametri n määrää funktion epälineaarisuusasteen: suuremmilla n:n arvoilla polulla tarvitsee olla vain hieman enemmän feromonia kuin vaihtoehdollaan, jotta valinta painottuisi todennäköisesti tätä kohti. Säätöparametreille tavataan antaa arvot kokeellisten tulosten perusteella. Deneubourg ja muut [1990] esittävät parametreille parhaiten sopiviksi arvoja $n \approx 2$, $k \approx 20$. Mikäli

tällöin A_i on merkittävästi suurempi kuin B_i ja A_i on merkittävästi suurempi kuin 20, niin $P_A \approx 1$. Jos taas A_i on merkittävästi suurempi kuin B_i, mutta A_i on pienempi kuin 20, $P_A \approx 0.5$. Vastaava pätee käänteisesti myös P_B:lle. Kaavoista (3) ja (4) seuraa, että

$$A_{i+1} = \begin{cases} A_i + 1, \ jos\ \delta \leq P_A \\ A_i, \ jos\ \delta > P_A \end{cases}, \tag{5}$$

$$B_{i+1} = \begin{cases} B_i + 1, \ jos\ \delta > P_A \\ B_i, \ jos\ \delta \leq P_A \end{cases}, \tag{6}$$

$$A_i + B_i = i, \tag{7}$$

missä δ on tasajakautunut satunnaismuuttuja lukuväliltä $[0, 1]$ [Bonabeau et al., 1999].

4.1.1. Metaheuristiikka

Muurahaisyhdyskuntaoptimoinnin on muodostanut metaheuristiikaksi Dorigo kollegoineen [1999]. Muurahaisyhdyskuntaoptimointimetaheuristiikka käsittää korkean tason kuvauksen useimmille muurahaisyhdyskuntaoptimointialgoritmeille kombinatoristen optimointiongelmien ratkaisemiseksi.

Lähtötilanne optimointiongelmaa ratkaistaessa muurahaisyhdyskuntaoptimointi-metaheuristiikassa on, että tarvitaan:

1. Äärellinen ratkaisukomponenttien joukko C, ja
2. *Feromonimalli* T.

Joukon C alkioista muodostuu ongelman ratkaisu ja T muodostaa parametrisoidun todennäköisyysmallin, jonka avulla ratkaisu rakennetaan. Feromonimallin alkiot $t_i \in T$ liitetään yleensä joihinkin ratkaisukomponentteihin joukosta C, josta mallia hyödyntämällä poimitaan ratkaisukomponentteja. Näin muodostuu sekvenssi, joka on ratkaisu käsillä olevaan optimointiongelmaan. Metaheuristiikan ratkaisumalli on kaksiosainen iteratiivinen prosessi:

4. Muodostetaan ongelman ratkaisukandidaatteja käyttäen hyväksi feromonimallia, ja
5. Päivitetään feromonimallia käyttäen hyväksi löydettyjä ratkaisukandidaatteja siten, että seuraavilla iteraatioilla saatavista kandidaateista jalostuu parempia ratkaisuja. Oletetaan, että hyvät ratkaisut koostuvat hyvistä ratkaisukandidaateista. [Blum and Li, 2008]

Metaheuristiikka on esitelty korkean tason pseudokoodina algoritmissa 2 [Blum, 2005].

Algoritmi Muurahaisyhdyskuntaoptimointi
    while lopetusehto ei voimassa **do**
        $RakennaRatkaisu()$
        $PäivitäFeromoniJäljet()$
        $MuutToiminnot()$
    end
end Muurahaisyhdyskuntaoptimointi

Algoritmi 2. Korkean tason pseudokoodi muurahaisyhdyskuntaoptimointimetaheuristiikasta.

Algoritmin osa rakennaRatkaisu() voidaan esittää algoritmin 3 mukaisesti [Blum, 2005].

Algoritmi rakennaRatkaisu
    määrittele $N(s)$
    $s = \langle\,\rangle$
    while $N(s) \neq \emptyset$ **do**
        $c = ValitseJoukosta(N(s))$
        $s = s.append(c)$
        määrittele $N(s)$
    end
end

Algoritmi 3. Pseudokooditarkennus ratkaisun rakentamisesta muurahaisyhdyskuntaoptimoinnissa.

Algoritmissa ratkaisu $C = \{c_i, \dots, c_n\}$ määritellään käsiteltävän ongelman perusteella. Ratkaisun rakentaminen aloitetaan tyhjästä ratkaisusta $s = \langle\,\rangle$ ja ratkaisumenetelmän suorituksen kulloisellakin iteraatiolla sekvenssiin lisätään komponentti joukosta $N(s) \subseteq C \setminus s$. Ratkaisukomponentin valinta joukosta $N(s)$ tehdään stokastisesti perustuen käytössä olevaan feromonimalliin. Todennäköisyysperustetta, jolla valinta tehdään, kutsutaan *transitiotodennäköisyydeksi* (transition probability) ja se määräytyy useimmissa muurahaisyhdyskuntaoptimointimetaheuristiikkaan perustuvissa algoritmeissa kaavalla [Blum, 2005]

$$p(c_i|s) = \frac{[\tau_i]^\alpha * [\eta(c_i)]^\beta}{\sum_{c_j \in N(s)} [\tau_j]^\alpha * [\eta(c_i)]^\beta}, \forall c_i \in N(s) \qquad (8)$$

missä η on valinnainen painotusfunktio, joka antaa heuristisen arvon $\eta(c_j)$ vaihtoehtoisille komponenteille $c_j \in N(s)$ ja eksponentit α ja β ovat painotusparametreja, $\alpha, \beta > 0$, joiden avulla voidaan kontrolloida, missä määrin funktiossa ovat painotettuina feromonimallin määräämä feromoni-informaatio $t_i \in T$ ja funktion η määräämä heuristinen informaatio.

Ratkaisun rakentamisen jälkeen muurahaisyhdyskuntaoptimointimetaheuristiikassa päivitetään feromonimalli. Mallin päivityksellä on kaksi komponenttia: feromonien haihdutus ja vahvistus. Feromonien haihdutuksessa vähennetään tasaisesti kaikkien feromonipainojen arvoa, millä vältetään liian aikainen konvergenssi hakuavaruuden osiin, jotka vaikuttavat menettelyn suorituksen alkuvaiheessa lupaavilta ratkaisukandidaattien löytämisen kannalta. Feromonien vahvistuksessa lisätään löydettyihin hyviin ratkaisuihin liittyvien komponenttien feromoniarvoja. Feromonimallia päivitetään kaavalla

$$\tau_i = (1-\rho) * \tau_i + \rho * \sum_{\{s \in S_{upd} | c_i \in s\}} w_s * F(s), i = 1, \dots, n, \qquad (9)$$

missä S_{upd} on päivityksessä käytettävien ratkaisukandidaattien joukko, $\rho \in (0,1]$ on feromonin haihtumistahtia kuvaava parametri ja $F: S \to \mathbb{R}^+$ on laatufunktio, jolla pätee $f(s) < f(s') \Rightarrow F(s) \geq F(s'), \forall s \neq s' \in S$, missä $f(s): S \to \mathbb{R}^+$ on optimoitava tavoitefunktio. Päivityksessä käytettyjen ratkaisuiden joukko S_{upd} vaihtelee toteutusten välillä. Se koostuu yleensä kulloisellakin iteraatioilla rakennettujen ratkaisuiden joukosta sekä parhaasta toistaiseksi tunnetusta ratkaisusta. Metaheuristiikan eri algoritmivariantit eroavat toisistaan lähinnä metodissa, jolla ne päivittävät feromonimalliaan. [Blum, 2005]

Metaheuristiikkakuvauksen valinnainen proseduuri muutToiminnot käsittää toimintoja, joita yksittäiset muurahaiset eivät voi toteuttaa, mutta joilla voidaan keskitetysti kontrolloida hakusuoritusta, esimerkiksi vähentää tai lisätä ylimääräisiä feromonipainotuksia [Blum, 2005].

4.1.2. Ant System

Ensimmäinen muurahaisyhdyskuntaoptimointimetaheuristiikkaan pohjaava algoritmi konkreettisen ongelman ratkaisemiseksi oli Dorigon kehittämä [1992] Ant System. Se on heuristinen algoritmi kauppamatkustajan ongelman ratkaisemiseksi approksimoimalla.

Ant Systemissä muurahaispopulaation jokainen jäsen generoi ratkaisun kauppamatkustajan ongelmaan kulkemalla Hamiltonin syklin graafin läpi, valitsemalla kulloisessakin rakennusvaiheessa seuraavan vierailtavan solmun käyttäen hyväksi stokastista *tilasiirtymäsääntöä* (state transition rule), joka suosii kaaria, joilla on pienet painotukset etsien näin lyhyitä reittejä kaupunkien välillä. Stokastisuudella tarkoitetaan tässä todennäköisyysmalliin perustuvaa valintaa. Kun jokainen muurahaispopulaation jäsen on muodostanut ratkaisukandidaatin, päivittävät muurahaiset graafin kaarille asetetut feromoniarvot käyttäen hyväksi *feromonipäivityssääntöä* (pheromone updating

rule), minkä seurauksena kaikkien kaarien feromoniarvoja vähennetään hieman. Tämän jälkeen muurahaiset kasvattavat feromoniarvoja kaarilla, jotka kuuluvat kunkin muurahaisen löytämään sykliin. Feromoniarvojen kasvun määrä on suhteessa löydetyn ratkaisukandidaatin hyvyyteen – mikäli syklin kokonaispituus on pieni, kasvatetaan tälle kuuluvien kaarien feromoniarvoja enemmän. Algoritmi perustuu näiden kahden suoritusvaiheen iterointiin. [Dorigo and Gambardelle, 1997]

Algoritmin käyttämä tilasiirtymäsääntö on

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]\cdot[\eta(r,s)]^\beta}{\sum_{u\in J_k(r)}[\tau(r,u)]\cdot[\eta(r,u)]^\beta}, & s \in J_k(r) \\ 0, & s \notin J_k(r) \end{cases}, \qquad (10)$$

missä $p_k(r,s)$ ilmaisee solmussa r sijaitsevan muurahaisen k todennäköisyyden valita solmu s. Kaaren (r,s) feromoniarvon ilmaisee τ ja η ilmaisee kaaren painotuksen käänteisenä, eli $\eta = \frac{1}{\delta}$ ja $\delta(r,s)$ on siis kaaren (r,s) painotus, eli kaupunkien r ja s välinen etäisyys. Parametri $\beta > 0$ on säätöparametri, jolla kontrolloidaan sitä, missä suhteessa feromoniarvojen ja kaarien painotukset ovat tärkeydessä reittiä valittaessa. Joukko $J_k(r)$ on niiden solmuun r liittyvien solmujen joukko, joissa muurahainen k ei ole vielä vieraillut. Kaavan toimintaperiaatteena siis on, että reitinvalinnassa suositaan sellaisia kaaria, joilla on mahdollisimman pieni painotus ja joihin liittyy mahdollisimman suuri feromoniarvo. [Dorigo and Gambardelle, 1997]

Feromoniarvojen päivityssääntö toteutetaan Ant Systemissä seuraavasti:

$$\tau(r,s) = (1-p) * \tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k(r,s) \qquad (11)$$

ja

$$\Delta\tau_k(r,s) = \begin{cases} \frac{1}{L_k}, & (r,s) \in T_k \\ 0, & (r,s) \notin T_k \end{cases}, \qquad (12)$$

missä L_k on muurahaisen k löytämän ratkaisukandidaatin kokonaispituus, m on algoritmin suorituksessa käytettävän muurahaispopulaation jäsenten kokonaismäärä, p on feromonin haihtumista ohjaava säätöparametri väliltä $]0,1[$ ja T_k on muurahaisen k löytämä ratkaisukandidaatti. Kaavat (11) ja (12) pyrkivät simuloimaan samanaikaisesti luonnossa tapahtuvaa feromonin haihtumista ja reiteillä kulkevien muurahaisten feromonin eritystä. Yhdessä nämä mekanismit toimivat positiivisen vahvistamisen mallina ratkaisua etsittäessä: lyhyille reiteille kertyy enemmän feromonia, joka taasen huomioidaan algoritmin seuraavilla iteraatioilla siten, että kaavan (10) mukaisessa tilasiirtymäsäännössä muurahaiset valitsevat jatkossa todennäköisemmin lyhempiä reittejä. Feromonimalli simuloi siis luonnossa tapahtuvaa muurahaiskolonioiden sisäistä stigmergistä kommunikaatioita. [Dorigo and Gambardelle, 1997]

Ant System oli ensimmäisenä muurahaisyhdyskuntaoptimointialgoritmina uraauurtava menetelmä, mutta oli jo julkaisunsa aikoina suorituskyvyltään heikompi kuin aikansa parhaat tunnetut tekniikat kombinatoristen optimointiongelmien ratkaisuun. Kun käsiteltävän kauppamatkustajan ongelman tapauksen koko kasvoi tietyn rajapisteen, suunnilleen $n \approx 30$, yli, alkoi Ant Systemin ratkaisun löytämiseen kuluva aika kasvaa hyvin nopeasti epäkäytännöllisen suureksi. Sittemmin muurahaisyhdyskuntaoptimointialgoritmeja on kuitenkin tutkittu paljon ja Ant Systemin seuraajiksi on syntynyt kilpailukykyisempiä variantteja. [Dorigo and Stützle, 2004]

4.1.3. Variantteja

Ensimmäinen alkuperäistä Ant System -algoritmia tehostava variantti menettelystä oli Elitist Ant System. Parannuksena alkuperäiseen Ant System -algoritmiin on ajatus siitä, että korostettaisiin entisestään parhaan tunnetun ratkaisun merkitystä. Tämä korostus tapahtuu feromoniarvojen päivityksen yhteydessä kasvattamalla feromoniarvoja enemmän niillä kaarilla, jotka kuuluvat parhaaseen tunnettuun ratkaisuun. Feromonin päivityssääntö saa tässä muodon

$$\tau(r,s) = (1-p) * \tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k(r,s) + e\Delta_{bs}(r,s), \qquad (13)$$

ja

$$\Delta\tau_{bs}(r,s) = \begin{cases} \frac{1}{L_{bs}}, & (r,s) \in T_{bs} \\ 0, & (r,s) \notin T_{bs} \end{cases}, \qquad (14)$$

missä e on säätöparametri, jolla ohjataan parhaalle tunnetulle ratkaisulle annettua painoarvoa, L_{bs} on parhaan tunnetun ratkaisun yhteenlaskettu kokonaispituus ja T_{bs} on paras toistaiseksi tunnettu ratkaisu. Kokeellisesti on osoitettu, että valitsemalla sopiva säätöparametri e saadaan Elitist Ant Systemillä parempia tuloksia pienemällä suoritusiteraatiomäärällä kuin alkuperäisellä Ant Systemillä. [Dorigo and Stützle, 2004]

Toinen hyväksi havaittu parannus alkuperäiseen algoritmiin on Bullnheimerin kollegoineen [1997] kehittämä Rank-Based Ant System. Rank-Based Ant Systemissä hyödynnetään Elitist Ant Systemin mekanismia, jossa parhaan tunnetun ratkaisun merkitystä korostetaan lisäferomonilla. Tämän lisäksi feromonin päivitykseen sisällytetään mekanismi, jossa muurahaisten erittämän feromonin määrä on suhteessa näille löydettyjen ratkaisuiden pituuden määrittämään keskinäiseen järjestykseen. Ennen feromonin päivityssäännön soveltamista muurahaiset järjestetään nousevasti näiden löytämän ratkaisun hyvyyden perusteella ja näille annetaan järjestysnumero r. Jokaisella iteraatiolla $w-1$ parasta ratkaisua vaikuttavat feromonin päivitykseen, missä w on parhaalle ratkaisulle annettu painotus. Feromonin päivityssääntö saa siis muodon

$$\tau(i,j) = (1-p) * \tau(i,j) + \sum_{r=1}^{w-1} (\mathrm{w} - \mathrm{r})\Delta\tau_r(i,j) + w\Delta_{bs}(i,j), \qquad (15)$$

missä $\Delta\tau_r(i,j)$ ja $\Delta_{bs}(i,j)$ määritellään vastaavasti kuten muissa Ant System -algoritmeissa. Kaavassa (15) on käytetty solmujen välisiä kaaria ilmaisevina symboleina selkeyden vuoksi r:n ja s:n sijasta merkintöjä i ja j. On osoitettu kokeellisesti, että Rank Based Ant System on ongelmanratkaisussa huomattavasti parempi kuin alkuperäinen Ant System ja jossain määrin parempi kuin Elitist Ant System. [Bullnheimer et al., 1997]

Kolmas hyvinkin huomionarvoinen variantti on Stützlen ja Hoosin [2000] $MAX - MIN$ Ant System, joka on myös yksi tutkituimmista ja menestyneimmistä Ant System -varianteista [Blum, 2005]. Siinä esitetään neljä parannusta alkuperäiseen Ant Systemiin nähden:

1. Ainoastaan kulloisellakin iteraatiolla parhaan ratkaisun löytänyt muurahainen päivittää feromoniarvoja.

2. Mahdolliset feromoniarvot rajoitetaan välille $[\tau_{min,}, \tau_{min}]$. Tällä vältetään parhaan löydetyn ratkaisun ylikorostuminen.

3. Feromoniarvot alustetaan määritellyn rajan ylärajalle, mikä saa aikaiseksi sen, että mahdollisia reittejä tutkitaan enemmän suorituksen alkuvaiheessa, kunhan feromonin haihtumistahti on riittävän pieni.

4. Kun algoritmin suorituksessa havaitaan pysähtyneisyyttä, feromoniarvot alustetaan uudelleen etsintäprosessin elvyttämiseksi. Pysähtyneisyydeksi katsotaan tilanne, jossa on käyty jonkin raja-arvon mukainen määrä suoritusiteraatioita siten, ettei löydetty ratkaisu ole parantunut.

Feromonin päivityssääntö muotoutuu kaavaksi

$$\tau(i,j) = (1-\alpha) * \tau(i,j) + \Delta\tau_{best}(i,j), \qquad (16)$$

missä $\Delta\tau_{best}(i,j) = 1/L_{best}$. Implementaatiosta riippuen feromoniarvoja päivittävä muurahainen voi olla globaalisti parhaan tähän mennessä tunnetun ratkaisun löytänyt muurahainen tai viimeisimmällä iteraatiolla parhaan ratkaisun löytänyt muurahainen. Jälkimmäisessä tapauksessa $\Delta\tau_{best}(i,j) = 1/L_{ib}$, missä L_{ib} merkitsee viimeisellä iteraatiolla parhaan löydetyn ratkaisun kokonaispituutta. Kokeelliset tulokset osoittavat, että pienillä ongelmatapauksilla iteraation parhaan, ja vastaavasti suuremmilla ongelmainstansseilla globaalisti parhaan, löydetyn ratkaisun korostus johtaa parempiin lopputuloksiin. [Stützle and Hoos. 2000]

Feromoniarvojen rajauksella pyritään välttämään pysähtyneisyyttä etsinnässä. Voidaan osoittaa, että feromoniarvojen yläraja on rajattu funktion $1/pL_*$ mukaisesti, missä L_* on optimaalisen ratkaisun

kokonaispituus ja p feromonin haihtumista ohjaava parametri. Tätä ylärajaa estimoidaan funktiolla $1/L_{bs}$, missä L_{bs} on parhaan tähän mennessä löydetyn ratkaisun kokonaispituus. Arviota käytetään sitten hyväksi siten, että se asetetaan feromoniarvojen ylärajaksi τ_{max} aina kun löydetään uusi paras tunnettu ratkaisu. Feromonin alarajaksi τ_{min} asetetaan arvo τ_{max}/a, missä a on jokin säätöparametri. Koska $MAX - MIN$ Ant Systemin suorituksen alkuvaiheessa feromoniarvojen yläraja asetetaan suureksi, eroavat kaarille asetetut feromoniarvot toisistaan jokseenkin hitaasti. Tästä seuraa, että etsintä on hajautettua, eli hakuavaruutta tutkitaan kauttaaltaan. Tämän lisäksi hajauttamisvaikutusta vahvistaa feromoniarvoja uudelleenalustava mekanismi. [Dorigo and Stützle, 2004]

Yhtenä menestyneenä varianttina voidaan mainita Dorigon ja Gambardellen [1997] kehittämä Ant Colony System. Siinä varioidaan alkuperäistä algoritmia seuraavilta osin:

1. ACS hyödyntää muurahaisten kerryttämää informaatiota tehokkaammin tilasiirtymäsääntönsä avulla,

2. Feromonia haihtuu ja lisääntyy ainoastaan kaarilla, jotka kuuluvat parhaaseen toistaiseksi tunnettuun ratkaisuun, ja

3. Muurahaisten liikkuessa kaarilla, ne haihduttavat feromonia haun hajauttamiseksi.

ACS:n tilasiirtymäsääntöä kutsutaan *pseudosatunnaiseksi suhteellisuussäännöksi* (pseudo-random proportional rule) ja se on muotoa

$$j = \begin{cases} argmax_{\{l \in J_k(i)\}}\{[\tau(i,l)] * [\eta(i,l)]^{\beta}\}, & q \leq q_0 \\ J, & q > q_0 \end{cases}, \qquad (17)$$

joka kuvaa solmussa i sijaitsevan muurahaisen k seuraavan solmun j valintaa. Joukko $J_k(i)$ on tämän muurahaisen ratkaisun kannalta validi solmunaapurusto, q on satunnaismuuttuja tasaisesti jakautuneelta lukuväliltä [0,1], q_o on säätöparametri samalta lukuväliltä ja J on satunnaismuuttuja, jonka arvo valitaan kaavan (10) mukaisesti, käyttäen Ant System -algoritmin tilasiirtymäsääntöä, jossa valitaan säätöparametri $\alpha = 1$. Pseudosatunnainen suhteellisuussääntö suosii alkuperäisen tilasiirtymäsäännön kaltaisesti lyhyitä kaaria ja suuria feromoniarvoja. Säätöparametri q_o määrää missä suhteessa hajauttaminen ja tehostaminen huomioidaan: mikäli $q \leq q_o$, valitaan tehostamista korostava sääntö, toisaalta jos $q > q_0$ valitaan solmu enemmän hajauttamisvaikutusta korostavan tilasiirtymäsäännön perusteella. [Dorigo and Gambardelle, 1997]

ACS:n feromonin päivityssäännössä vain parhaaseen tunnettuun ratkaisuun liittyviä feromoniarvoja päivitetään. Päivityssääntö on muotoa

$$\tau(i,j) = (1-p) * \tau(i,j) + p\Delta\tau_{bs}(i,j), \forall (i,j) \in T_{bs}, \qquad (18)$$

missä $\Delta\tau_{bs}(i,j) = 1/L_{bs}$. Rajoittamalla feromonin päivitys vain yhteen ratkaisuun, muuttuu myös operaation aikakompleksisuus neliöllisestä lineaariseksi. Kaavan (18) esittämän feromonin päivityssäännön lisäksi ACS:ssä hyödynnetään paikallista feromonin päivitystä, joka suoritetaan muurahaisten liikkuessa kaarilla. Kun muurahainen on liikkunut kaarella (i,j), päivitetään kaaren feromoniarvot seuraavasti:

$$\tau(i,j) = (1-\xi) * \tau(i,j) + \xi\tau_0, \tag{19}$$

missä $\xi \in]0,1[$ ja τ_o ovat säätöparametreja ja parametri τ_0 on sama kuin feromonien alkuarvo. Kokeellisesti on havaittu, että hyvät arvot näille parametreille ovat $\xi = 0.1$ ja $\tau_o = 1/nT_{nn}$, missä n on ongelmainstanssin solmujen määrä ja T_{nn} on optimiratkaisun estimaattina lähimmän naapurin heuristiikalla saavutetun ratkaisun kokonaispituus. [Dorigo and Gambardelle, 1997]

Muurahaisyhdyskuntaoptimointialgoritmit lukeutuvat parhaiden tunnettujen ratkaisualgoritmien joukkoon useiden kombinatoristen optimointiongelmien ratkaisemiseksi [Blum, 2005]. Esitellyistä Ant System -varianteista jokainen suoriutuu tehtävästään paremmin kuin alkuperäinen algoritmi: kokeellisissa testauksissa on osoitettu näiden löytävän järjestelmällisesti parempia ratkaisuja kuin Ant System keskimäärin. Varianttien joukosta Ant Colony System löytää hyviä ratkaisuja lyhimmissä suoritusajoissa, kun taas $MAX-MIN$ Ant System käyttäytyy suorituksessaan siten, että vaikka algoritmi löytää suorituksensa alkuvaiheissa verrattain huonolaatuisia ratkaisuja, on sen löytämä lopullinen ratkaisu pääsääntöisesti laadukkain verrokkialgoritmiensa joukosta. Tutkimuksissa on järjestelmällisesti havaittu, että Ant Systemin varianteista parhaiten suoriutuvia ovat $MAX-MIN$ Ant System ja Ant Colony System. Stützlen ja Hoosin [2000] koeasetelmien perusteella $MAX-MIN$ Ant System on useimmissa ongelmainstansseissa Ant Colony Systemiä suorituskykyisempi algoritmi. Rank-Based Ant Systemin on näiden kahden jälkeen suorituskykyisimpien varianttien joukossa. Algoritmien suorituskykyä voidaan perusmuodostaan parantaa entisestään hybridisoimalla näitä perinteisten heuristiikkojen kanssa: yhdistämällä paikallisen haun menettelyitä muurahaisyhdyskuntaoptimointialgoritmeihin näistä saadaan entistä kilpailukykyisempiä ratkaisumenetelmiä. [Dorigo and Stützle, 2004]

4.2. Partikkeliparvioptimointi

Partikkeliparvioptimointi (Particle swarm optimization) on alun perin Kennedyn ja Eberhartin [1995] kehittämä parviälykkyysmetaheuristiikka. Se on muurahaisyhdyskuntaoptimoinnin kaltaisesti populaatioperustainen stokastinen optimointiheuristiikka, joka perustuu luonnossa havaittuihin mekanismeihin. Partikkeliparvioptimoinnissa mallinnetaan luonnossa muun muassa lintu- ja kalaparvissa esiintyvään parveutumiskäyttäytymiseen. Mallinnusta käytetään pohjana metaheuristiselle algoritmille jatkuvien optimointiongelmien ratkaisemiseen. Partikkeliparvioptimointia on sen esittelyn jälkeen tutkittu paljon ja se on tehokkaana ja sovelluskykyisenä menetelmänä optimointiheuristiikkojen suorituskykymittauksissa yksi tärkeimmistä vertailukohteista [Civicioglu and Besdok, 2011].

Partikkeliparvioptimoinnissa mallinnetaan parveilukäyttäytymistä käyttäen perusyksiköinä hakuavaruudessa liikkuvia partikkeleita, jotka vastaavat optimointiongelman ratkaisukandidaatteja. Partikkeleilla on sijainti, suunta ja nopeus, joiden mukaan ne liikkuvat ja yhdessä vuorovaikutuksen naapurustonsa muiden partikkelien kanssa yrittävät lähestyä parhaita tunnettuja sijainteja. Partikkeli valittiin alkuperäistutkimuksessa kompromissina yhtenäisen käsitteistön ylläpitämiseksi, vaikka partikkeleilla onkin vain kuvitteellinen massa ja tilavuus, sillä liikenopeuden ja kiihtyvyyden käsitteiden ajateltiin soveltuvan paremmin partikkeleille kuin pisteille. Parven käsitteen alkuperä partikkeliparvioptimoinnissa on Millonaksen [1993] esittelemät parven ja parviälyn pohjaperiaatteet. [Kennedy and Eberhart, 1995]

Partikkeliparvioptimointi liittyy läheisesti keinotekoisen elämän malleihin, joita on tutkittu kirjallisuudessa aikaisemmin: eläinten, esimerkiksi lintujen, parveilukäyttäytymistä oli tutkittu ja mallinnettu muun muassa grafiikkatutkimuksen kontekstissa [Reynolds, 1987]. Lintujen parveilukäyttäytymistä tutkiessa havaittiin, että parveilun avainmekanismeja on parven jäsenten paikalliset vuorovaikutukset, mikä on myös partikkelipariopimoinnin pohjana olevassa mallissa parvikäyttäytymisen perusmekanismi. Partikkeliparvioptimointi on osaltaan verrattavissa myös soluautomaatteihin siten, että partikkelit voidaan mieltää soluautomaatin solujen kaltaisiksi yksiköiksi, joita päivitetään samanaikaisesti samojen sääntöjen perusteella uusiin tiloihin ja prosessia iteroidaan, mistä syntyy kollektiivisesti eräänlaista ryhmäkäyttäytymistä.

Partikkeliparvioptimoinnin partikkelit päivittävät sijaintiaan iteratiivisesti seuraten hyviä, löydettyjä ratkaisukandidaatteja: partikkelit hakeutuvat kohti parasta tuntemaansa ratkaisua sekä parasta niiden naapuruston tuntemaa ratkaisua. Jokainen partikkeli siis etsii ratkaisuja avaruudesta, jonka määrittävät nämä kaksi pistettä. Partikkelien sijainnin päivitys tapahtuu seuraavasti:

$$\boldsymbol{v}_i = \boldsymbol{v}_i + \varphi_1 \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + \varphi_2 \otimes (\boldsymbol{p_g} - \boldsymbol{x}_i), \tag{20}$$

$$\boldsymbol{x_i} = \boldsymbol{x}_i + \boldsymbol{v}_i, \tag{21}$$

missä $\boldsymbol{v_i}, i = 1, \dots, n,$ merkitsee partikkeliparven $i{:}\,n$ partikkelin liikenopeusvektoria, $\boldsymbol{x}_i$ vastaavasti sen sijaintia, $\boldsymbol{p}_i$ on kyseisen partikkelin paras tähän asti löytämä sijainti ja $\boldsymbol{p}_g$ on paras sijainti sen naapurustossa. Lisäksi $\varphi_1 = c_1\boldsymbol{R}_1$ ja $\varphi_2 = c_2\boldsymbol{R}_2$, missä $\boldsymbol{R}_1$ ja $\boldsymbol{R}_2$ ovat funktioita, jotka palauttavat arvoinaan lukuväliltä $[0,1]$ poimittuja satunnaislukuja sisältäviä vektoreita. Parametrit c_1 ja c_2 ovat kiihtyvyyskertoimina toimivia säätöparametreja. Operaattori $\otimes$ merkitsee alkioittain tehtävää vektorikertolaskua. Partikkelin liikenopeus $\boldsymbol{v}_i$ koostuu kaavan (20) mukaisesti kolmesta osasta: liikemääräosasta, kognitiivisesta osasta ja sosiaalisesta osasta. Liikemääräosa $\boldsymbol{v}_i$ on partikkelin liikenopeus edellisellä iteraatiolla ja se ohjaa partikkelin liikettä suuntaan, johon se oli aikaisemmin matkalla. Kognitiivinen osa $\varphi_1 \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i)$ ohjaa partikkelia kohti parhaita sijainteja, joita tämä on hakuavaruutta kartoittaessaan kohdannut. Viimeinen, eli sosiaalinen osa, $\varphi_2 \otimes (\boldsymbol{p_g} - \boldsymbol{x}_i)$, ohjaa partikkelia kohti parhaita parven löytämiä sijainteja. Sosiaalisen osan määrittelyssä käytetyn termin $\boldsymbol{p}_g$ määrittely riippuu implementaatiossa käytettävästä naapurustotopologiasta. Esimerkkeinä

vaihtoehtoisista naapurustotopologioista on esitelty kirjallisuudessa kehä, tähti ja von Neumann -topologiat. Partikkeliparvioptimoinnin implementaatiot voidaan jakaa kahteen ryhmään sen perusteella, käyttävätkö ne ylipäätään naapurustopologiaa hyväkseen rajoittamassa parhaan tunnetun ratkaisun valintaa: naapurustorajoitetta hyväksikäyttäviä partikkeliparvioptimointialgoritmeja kutsutaan $lbest$-partikkeliparvioptimointialgoritmeiksi, kun taas algoritmeja, jotka valitsevat parhaan ratkaisukandidaatin koko parven joukosta, kutsutaan $gbest$-partikkeliparvioptimointialgoritmeiksi. [Blum and Li, 2008]

Kaavassa (20) on huomionarvoista, että liikenopeus saattaa helposti kasvaa liian suureksi, mikäli partikkeli on riittävän kaukana suuntamerkkeinä käytettävistä sijainneista $\boldsymbol{p}_i$ ja $\boldsymbol{p}_g$, kun partikkeli liikkuu käsiteltävän hakuavaruuden ulkopuolelle. Tämän rajoittamiseksi partikkeliparvioptimoinnissa voidaan liikenopeustekijöiden saamat arvot rajoittaa halutulle arvoalueelle esittelemällä rajaparametri V_{max}. Tämä ei välttämättä estä partikkelien päätymistä hakuavaruuden ulkopuolelle, mutta koska se rajoittaa partikkelien yksittäisten askelten mittaa, voidaan sillä hallita partikkelien keskinäistä hajaantumista. Parametrin arvon valinta vaikuttaa hajauttamisen ja tehostamisen väliseen tasapainoon, joten optimaalisen arvon valinta ei ole suoraviivaista. Rajaparametrin V_{max} optimaalinen arvo riippuu käsiteltävästä optimointiongelmasta, eikä siihen tunneta selkeää valintasääntöä. Tämän lisäksi liikenopeuksien rajoittaminen saattaa hankaloittaa partikkelin liikeradan konvergenssia kohti ratkaisua. [Poli et al., 2007]

Erillisen liikerajaparametrin V_{max} sijasta partikkelien hajautumista voi hallita käyttämällä inertiakerroinparametria ω. Inertiapainotettuna partikkeliparvioptimointi ei tarvitse erikseen rajattua liikenopeuksien arvoaluetta parven konvergoitumista varten. Inertiakerroinparametri ohjaa partikkeliparven haun kulkua siten, että sen arvoilla $\omega < 1$, partikkelien liikenopeudet hidastuvat suorituksen edetessä. Mikäli sitä vastoin $\omega > 1$, kasvavat partikkelien liikenopeudet ja ne ajaantuvat ennen pitkää hakuavaruuden ulkopuolelle. Inertiapainotettuna liikenopeuden päivitys tehdään kaavalla

$$\boldsymbol{v}_i = \omega \boldsymbol{v}_i + \varphi_1 \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + \varphi_2 \otimes (\boldsymbol{p_g} - \boldsymbol{x}_i). \qquad (22)$$

Kertoimen arvoiksi suositellaan algoritmin suorituksen alkuaiheessa $\omega \approx 0.9$ ja tämän vähentämistä asteittain suorituksen edetessä kohti arvoa $\omega \approx 0.4$. [Eberhart and Shi, 2000]

Clerc [1999] on esitellyt, että konvergenssin takaamiseksi partikkeliparvioptimoinnissa olisi tarpeen käyttää partikkelien liikenopeuden päivityksen yhteydessä yleistä rajoitekerrointa. Yksinkertaisessa muodossa rajoitekerroinpäivitetty liikenopeuden päivityskaava on

$$\boldsymbol{v}_i = \chi(\boldsymbol{v}_i + \varphi_1 \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + \varphi_2 \otimes (\boldsymbol{p_g} - \boldsymbol{x}_i)), \qquad (23)$$

missä $\chi = \frac{2}{|2-\varphi-\sqrt{\varphi^2-4\varphi}|}$, jossa taasen $\varphi = c_1 + c_2, \varphi > 4$. Mikäli asetetaan $\varphi = 4.1, c_1 = c_2 = 2.05$, tulee rajoitekertoimen arvoksi $\chi = 0.7298$ ja täten partikkelin kunkin vaiheen liikenopeudeksi saadaan edellisen vaiheen liikenopeus skaalattuna $0.7298 * 2.05 \approx 1.496$. Tämä takaa Eberhartin ja

Shin [2000] mukaan parven konvergenssin ilman tarvetta inertiaparametrin käyttöön. On kuitenkin hyödyllistä ottaa yleisen rajoitekertoimen ohella käyttöön jokin menettely liikenopeuden rajoittamiseksi, sillä tästä on osoitettu olevan joidenkin ongelmien tapauksissa hyötyä. [Blum and Li, 2008]

Partikkeliparvioptimoinnin alkuperäisversion pseudokoodi on esitetty algoritmissa 4. Algoritmin lopetuskriteereinä käytetään yleensä joko iteraatiomaksimia tai riittävän pientä tavoitefunktion erotusta optimista

---

Algoritmi Partikkeliparvioptimointi
Syötteet: Satunnainen partikkeliparvi $\boldsymbol{x}_i$, $(i = 1, 2, \dots, n)$, tavoitefunktio $f(x)$.
Tuloste: Paras löydetty ratkaisu x_*

while $not\ lopetusKriteerit()$ **do**
for $i = 1, \dots, n$ **do**
if $f(\boldsymbol{x}_i) > f(\boldsymbol{p}_i)$ **then** $\boldsymbol{p}_i = \boldsymbol{x}_i$
$\boldsymbol{p_g}$ = max($N_i(p)$) // $N_i(p)$ ilmaisee naapuruston positiot.
$PäivitäLiikenopeus(i)$
$PäivitäSijainti(i)$
end for
end while
end Partikkeliparvioptimointi

Algoritmi 4: Partikkeliparvioptimoinnin pseudokoodi.

Eberhart ja Shi [2000] havaitsivat kuitenkin, että suositeltava menettely säätöparametrien käytössä on asettaa yleisen säätöparametrin ohella liikenopeuden yläraja-arvoksi $V_{max} = X_{max}$, missä X_{max} esittää dynaamista muuttujan arvoaluetta kutakin käsiteltävää ulottuvuutta kohden, jolloin vältetään ongelmakohtaiset säätöparametrit. Näillä säätöparametreilla viritetty partikkeliparvioptimointi on myös tällä hetkellä kyseisen metaheuristiikan kanoninen sovellutus. On myös huomionarvoista, että inertiakertoimella ja yleisellä rajoitekertoimella varustetut algoritmit ovat keskenään matemaattisesti yhtäläiset. Inertiakerroinalgoritmi voidaan muuttaa yleiseen rajoitekerroinmuotoon transformaatiolla vastaavien säätöparametrien välillä. [Poli et al., 2007]

On huomionarvoista, että kaavan sijaintitermit $\boldsymbol{p_i}$ ja $\boldsymbol{p}_g$ voidaan yhdistää termiksi $\boldsymbol{p}$. Liikenopeuden päivitys tämän yhdistämisen jälkeen muotoa

$$\boldsymbol{v}_i = \boldsymbol{v}_i + \varphi \otimes (\mathbf{p} - \mathbf{x}_\mathrm{i}), \qquad (24)$$

missä $\boldsymbol{p} = \frac{\varphi_i \boldsymbol{p}_i + \varphi_2 \boldsymbol{p}_g}{\varphi_1 + \varphi_2}$, eli $\boldsymbol{p}_i$:n ja $\boldsymbol{p}_g$:n painotettu keskiarvo, ja $\varphi = \varphi_1 + \varphi_2$. Tällöin kaavan (24) mukaisesti partikkeliparven haku konvergoituu kohti termin $\boldsymbol{p}$ määrittelemää pistettä [Blum and Li, 2008]. Mendes kollegoineen [2004] on esittänyt termin jatkoyleistyksen partikkeliparvioptimoinnin

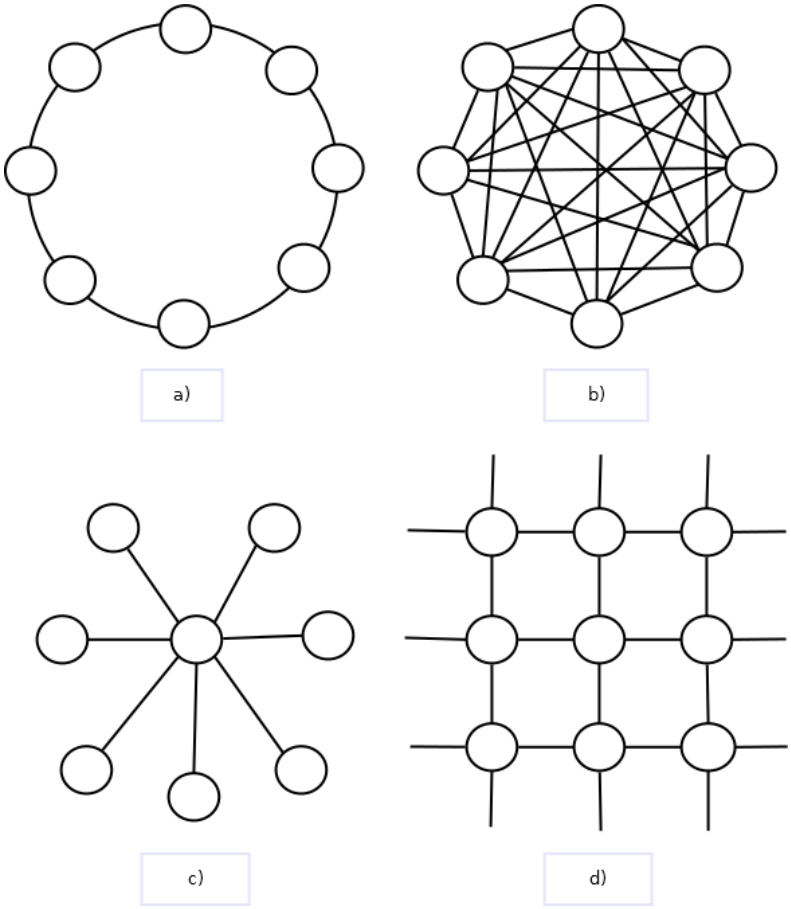
variantissa, jossa parvi on täysin informoitu. Mendesin partikkeliparvioptimoinnin versiossa termi $\boldsymbol{p}$ lasketaan kaavalla

$$\boldsymbol{p} = \frac{\sum_{k \in N} r \left[0, \frac{c_{max}}{|N|}\right] \otimes \boldsymbol{p}_k}{\sum_{k \in N} \varphi_k}, \tag{25}$$

missä N on partikkelin naapurusto, johon kulloinenkin partikkeli myös itse sisältyy ja $\boldsymbol{p}_k$ on partikkelin k löytämä edellinen paras sijainti. Kun kanonisessa partikkeliparvioptimoinnissa partikkelin sijaintiin vaikuttavat sen paras itse löytämä positio sekä partikkelin naapuruston paras sijainti, täysin informoidussa partikkeliparvioptimoinnissa partikkelia ohjaa sen koko naapurusto. Kun käytettävät parametrit valitaan oikein, tuottaa täysin informoitu partikkeliparvioptimointi parempia tuloksia pienemmillä iteraatiomäärillä kuin perinteinen algoritmi. Toisaalta täysin informoidussa partikkeliparvioptimoinnissa on käyttöön valitulla naapurustotopologialla huomattavasti suurempi merkitys kuin kanonisessa algoritmissa [Poli et al., 2007].

Yleisimmät käytetyt naapurustotopologiat on esitetty kuvassa 3. Kehätopologiassa kukin solmu on yhdistetty kahteen naapuriinsa. Tähtitopologiassa yksi keskeinen solmu on yhdistetty kaikkiin muihin solmuihin. Täysin yhdistetyssä topologiassa kaikki solmut ovat yhdistetty toisiinsa. Toisaalta von Neumann -topologiassa solmut ovat järjestetyt hilaan, jossa jokaisella solmulla on neljä naapuria: pohjoinen, eteläinen, itäinen ja läntinen naapuri. Kuvassa 3 listattujen topologioiden ohella myös satunnaisgeneroitu naapurustotopologia on yleisesti käytetty lähestymistapa. [Blum and Li, 2008]

Kanonisessa partikkeliparvioptimoinnissa naapurustotopologiat ovat staattisia, eivätkä siten muutu algoritmin suorituksen edetessä. Topologioihin liittyvien suorituskykyseikkojen kannalta saattaa kuitenkin olla hyödyllistä tutkia hakuavaruutta ensin rajoitettua naapurustotopologiaa hyväksikäyttäen ja vaihtaa strategiaa algoritmin suorituksen edetessä kohti laajempaa naapurustoa, sillä rajoitetummilla naapurustotopologioilla voidaan tutkia hakuavaruutta tehokkaammin ja laajemmilla naapurustoilla saavutetaan taasen tehokkaampi konvergenssikäyttäytyminen. Vaihtoehtoisesti naapurustotopologia voidaan generoida muutaman, esimerkiksi viiden, iteraation jälkeen satunnaisesti uudelleen – tällä tavoin on saatu hyviä tuloksia etenkin multimodaalisten optimointiongelmien ratkaisussa. On myös esitelty menettelyitä, joissa partikkeliparvelle muodostetaan partikkelien keskinen hierarkia, joissa parempia tuloksia saavat partikkelit saavat hierarkiassa korkeamman sijan ja hierarkiassa korkeammille partikkeleille annetaan suurempi merkitys positioita laskettaessa. [Poli et al., 2007]



Kuva 3. Yleisimpiä partikkeliparvioptimoinnin naapurustopologioita. Topologiat ovat: a) kehätopologia, b) täysin yhdistetty topologia, c) tähtitopologia ja d) von Neumann -topologia 2D-representaationa.

Partikkeliparvioptimoinnin partikkelien käyttäytymistä on tutkittu myös teoreettisesti, tarkastellen partikkelien liikeratojen käyttäytymistä algoritmin suorituksen aikana. Partikkelien liikeratojen käyttäytyminen on sidoksissa käytettyihin liikenopeuden päivityksen säätöparametreihin ja täten teoreettiset tutkimukset tarjoavat tuloksinaan myös heuristiikkoja parametrien oikeaan valintaan. Näiden heuristiikkojen perusteella voidaan säätöparametreille valita sopivat arvot siten, että taataan partikkeliparven liikeratojen konvergenssi kohti tasapainoalueita. Konvergenssilla kohti tasapainoaluetta tarkoitetaan, että tietyillä edellytyksillä algoritmin suoritus saavuttaa pisteen, jossa saavutetut arvot eivät enää muutu. Nämä tulokset eivät kuitenkaan takaa konvergenssia kohti hakuavaruuden optimia. [Bergh and Engelbrecht, 2006]

Partikkeliparvioptimointi lukeutuu eniten käsiteltyjen parviälykkyysheuristiikkojen joukkoon. Tässä ominaisuudessa tälle on kehitetty tutkimuksessa myös lukuisia variantteja, joista osa on yleisiä parannelmia kanoniseen optimointialgoritmiin ja osa muunnelmia, jotka tähtäävät suorituskyvyn parantamiseen tietyillä sovellusalueilla. Esimerkiksi multimodaalisten optimointiongelmien tehokkaammaksi ratkaisemiseksi on kehitetty moniparvimenetelmiä, joissa hakuavaruutta kartoittaa

pienten partikkeliparvien joukko [Liang and Suganthan, 2005]. Eräs mielenkiintoisimmista muunnelmakonsepteista on partikkeliparvioptimoinnin hyödyntäminen uusilla sovellusalueilla. Vaikka menetelmä kehitettiin alun perin nimenomaan jatkuvien optimointiongelmien ratkaisemiseksi, on tutkimuksessa kehitetty menetelmästä diskreettejä muunnelmia. Kennedy ja Eberhart [1997] esittelivät jo varhain yksinkertaisen muunnoksen algoritmista, jossa käsitellään reaalilukujen sijasta binäärisiä bittijonoja. Tässä muunnelmassa partikkelin liikenopeutta käytetään rajamääreenä, joka kontrolloi, arvioidaanko partikkeliparven komponentti binääriarvona 0 vai 1. Siinä liikenopeuden päivityssääntö pysyy samana kuin alkuperäisessä implementaatiossa, mutta p_{id} ja v_{id} rajoitetaan välille $[0, 1]$. Liikenopeuden arvoalueen rajoittamisessa hyödynnetään sigmoidifunktiota rajoittavana transformaationa seuraavasti

$$s(v_{id}) = \frac{1}{1 + \exp(-v_{id})}, \tag{26}$$

missä v_{id} on parven i:nnen partikkelin komponentti d. Bittijonon bittiä edustavan komponentin x_{id} arvo määräytyy vertaamalla sigmoidifunktion arvoa satunnaismuuttujaan $r \in [0,1]$:

$$x_{id} = \begin{cases} 1, & r < s(v_{id}) \\ 0, & r \geq s(v_{id}) \end{cases}. \tag{27}$$

Vertailtaessa geneettisiin optimointialgoritmeihin tämä binäärinen partikkeliparvioptimoinnin variantti on osoittautunut hyväksi menetelmäksi: koeasetelmien perusteella algoritmi saavuttaa usein optimiratkaisut. Lukuun ottamatta yksinkertaisimpia testiongelmia se saavuttaa paremman suorituskyvyn kuin geneettiset algoritmit [Poli et al., 2007].

Yksinkertaisen binäärisen variantin lisäksi partikkeliparvioptimointi on sovellettavissa myös lukuisten diskreettien ja hybridioptimointiongelmien ratkaisuun. Kirjallisuudessa on esitelty sovellutukset lukuisiin tunnettuihin kombinatoristen optimointiongelmien ratkaisemiseksi. Eräänä mielenkiintoisena sovellutuksena voidaan mainita esimerkiksi partikkeliparvioptimoinnin sovellutus kauppamatkustajan ongelman ratkaisemiseksi. Siinä määritellään partikkelien liikenopeus operaatioina, joilla vaihdetaan solmuja keskenään, näiden solmujen transpositioiden määritellen solmujoukosta saadun permutaation. Ratkaisussa määritellään sovellusongelman kanssa yhteensopiviksi muut tarvittavat operaatiot, mukaan lukien liikenopeuden negaatio, yhteen-, vähennys- ja kertolaskuoperaatiot liikenopeuden ja position suhteen sekä pisteiden välinen etäisyys. Operaatioiden sopivan määrittelyn myötä voidaan itse liikenopeuden ja sijainnin päivityksessä perustaa laskenta samoihin kaavoihin kuin kanonisessa partikkeliparvioptimoinnissa. Vaikka kaavojen operaatioiden merkitykset ovat dramaattisesti erilaiset kuin perinteisessä partikkeliparvioptimoinnissa, on prosessi näissä periaatteellisesti sama. Suorituskyvyltään kauppamatkustajan ongelman ratkaisuun tehty muunnelma partikkeliparvioptimoinnista on lupaava, joskin hyvien ratkaisuiden tuottamiseksi tämä tarvitsee hybridisointia muiden menettelyiden kanssa. Menettely on hyvä kartoittamaan lupaavia alueita hakuavaruudessa, mutta kartoitettujen alueiden

jalostamiseksi hyviksi ratkaisuiksi, kannattaa hyödyntää esimerkiksi paikallisen haun heuristiikan menetelmiä. [Clerc, 2010]

Toisenlaista muunnelmaa edustaa adaptiivinen partikkeliparvioptimointi, jonka lähtökohtaisena tavoitteena on toimia itseohjautuvana siten, että algoritmi etsisi itse arvot säätöparametreilleen, esimerkiksi partikkeliparven koolle, mutta olisi kuitenkin samalla suorituskyvyltään kilpailukykyinen. Clerc [2010] kehitti tätä tarkoitusta varten partikkeliparvioptimoinnin muunnelman *tribes*, jossa partikkeliparven koko muuttuu algoritmin suorituksen aikana suorituksesta saadun palautteen perusteella. Tribesissä partikkeliparvi jaetaan alipopulaatioihin, joita kutsutaan heimoiksi (engl. tribe), joilla on oma sisäinen rakenteensa. Hyviä ratkaisuja saavat heimot hyötyvät heikoimmin suoriutuvien jäsentensä poistamisesta parvesta, kun taas verrattain huonoja ratkaisuja löytävät heimot saattavat hyötyä uusista partikkeleista heimossa. Populaatiostruktuuri rakennetaan uudelleen joka $L/2$ iteraation välein, missä L on partikkelipopulaation linkkien määrä. Koeasetteluissa tribes on osoittautunut lupaavaksi optimointitekniikaksi ja suoriutuu testifunktioilla paikoin paremmin kuin alkuperäinen partikkeliparvioptimointi. Adaptiivisen menettelyn suurin etu on siinä, ettei optimointiongelmien ratkaisussa tarvita säätöparametrien asettamista, vaan riittää, että algoritmin soveltaja määrittelee ainoastaan hakuavaruuden, optimoitavan funktion, halutun tarkkuusasteen sekä iteraatiomaksimin. [Clerc, 2010]

4.3. Mehiläisyhdyskuntaoptimointialgoritmi

Mehiläisyhdyskuntaoptimointialgoritmi eli ABC-algoritmi (Artificial bee colony algorithm) on Karabogan ja Basturkin [2007] kehittämä jatkuvien optimointiongelmien ratkaisuun suunniteltu parviälykkyyttä hyödyntävä optimointimetaheuristiikka, joka perustuu mehiläisyhdyskuntien käyttäytymiseen ravinnon etsinnässä. Mehiläisten käyttäytymiseen perustuen on laadittu useita metaheuristisia algoritmeja, muun muassa Teodorovićin ja Dell'Orcon [2005] BCO (Bee Colony Optimization) sekä Driasin ja kollegoiden [2005] BSO (Bee Swarm Optimization) -metaheuristiikat determinististen kombinatoristen optimointiongelmien ratkaisemiseksi. Ensimmäinen esitelty mehiläisten käyttäytymiseen pohjaava heuristinen optimointialgoritmi oli Yangin [2005] virtuaalimehiläisalgoritmi. Huomionarvoisimpiin mehiläisten käyttäytymiseen perustuviin optimointialgoritmeihin lukeutuu Phamin ja kollegoiden [2006] mehiläisalgoritmi sekä Karabogan ja Basturkin [2007] mehiläisyhdyskuntaoptimointialgoritmi. Etenkin multimodaalisten optimointiongelmien ratkaisuun kehitetty mehiläisyhdyskuntaoptimointialgoritmi on saanut osakseen huomiota ja on mukana useissa optimointialgoritmien suorituskykyä mittaavissa vertailututkimuksissa [Civicioglu and Besdok 2011].

Mehiläisten ravinnonetsintäkäyttäytyminen luonnossa käsittää monien muiden parvieläinten tapaan parviälykkyyttä: mehiläisten keskinäisestä vuorovaikutuksesta verrattain yksinkertaisten mekanismien välityksellä parvessa syntyy kollektiivisesta toimintaa, joka osoittaa parven emergenttiä älykkyyttä. Mehiläiset jakavat yhdyskunnassa tehtäviä dynaamisesti ja reagoivat ympäristössään tapahtuviin muutoksiin, niillä on muisti, erinomainen astinta, ne huolehtivat parvensa

kuningattaresta sekä jälkeläisistään ja kommunikoivat tehokkaasti keskenään. Mehiläisyhdyskuntien toimintamekanismeihin lukeutuu useita eri järjestelmiin mallinnettavissa olevia toimintoja. Tutkimuksessa mallinnettujen mekanismien joukossa ovat esimerkiksi kommunikaatiomenetelmät, eli mehiläisten tanssi, ravinnonetsintä, tehtävien allokointi, kuningatarkäyttäytyminen, kollektiivinen päätöksenteko, pesäpaikan valintamekanismi, parittelu, pölyttäminen sekä navigointi. [Karaboka and Akay, 2009]

Mehiläisyhdyskuntaoptimointialgoritmissa mallinnettavana käyttäytymismekanismina on ravinnonetsintämekanismi. Luonnossa mehiläiset etsivät ravintoaan tunnetusti kasvien tarjoamasta nektarista, jota mehiläiset muuttavat hunajaksi. Karaboga [2005] on muodostanut mehiläisten ravinnonetsinnästä yksinkertaistetun mallin, joka koostuu kolmesta keskeisestä komponentista ja kahdesta käyttäytymismuodosta: ravinnonhakuun värvääminen sekä ravinnonlähteen hylkääminen. Mallin kolme pääkomponenttia ovat seuraavat.

1. Ravinnonlähde: mehiläiset arvottavat ravinnonlähteen arvon riippuen muun muassa sen etäisyydestä pesästä, hyödyntämisen helppoudesta ja sen sisältämän ravinnon laadusta. Mehiläisyhdyskuntaoptimoinnissa ravinnonlähteen arvoa kuvataan yhdellä suureella.

2. Aktiiviset ravinnonetsijät, jotka pitävät hallussaan tietoa käsittelemänsä ravinnonlähteestä. Tietoon lukeutuu lähteen etäisyys, sijainti ja käsiteltävyys. Aktiiviset ravinnonetsijät jakavat tätä informaatiota muun parven kanssa.

3. Passiiviset ravinnonetsijät, joiden tehtävänä on etsiä uusia hyödynnettäviä ravinnonlähteitä. Passiiviset ravinnonetsijät jaetaan kahteen luokkaan: tiedustelijoihin ja seuraajiin. Tiedustelijat tutkivat ympäristöään etsien uusia ravinnonlähteitä, kun taas seuraajat odottavat pesässä, tavoittaen ravinnonlähteitä muilta mehiläisiltä saadun, parvelle jaetun informaation perusteella.

Mehiläisten välinen kommunikaatio tapahtuu mehiläistanssin välityksellä. Mehiläisyhdyskunnalla on parven keskiselle kommunikaatiolle varattu tanssialueeksi kutsuttu sijainti. Tanssin muotoja on erilaista ja sen perusteella mehiläiset välittävät toisilleen informaation tanssivien mehiläisten löytämistä ravinnonlähteistä. Tanssia seuraavat mehiläiset valitsevat sitten tarjolla olevista ravinnonlähteistä parhaan, värväytyen näin kyseisen ravinnonlähteen hakuun. Koska aktiiviset ravinnonetsijät jakavat saavuttamansa informaation todennäköisyydellä, joka on suoraan verrannollinen suhteessa ravinnonlähteen laatua kuvaavaan suureeseen, koskee tanssialueen ravintolähdeinformaatiosta tämän seurauksena suurempi osa laadullisesti hyviä ravinnonlähteitä. [Karaboka, 2005]

Ravinnonetsintäprosessin alussa ravinnonetsijä on passiivinen. Siitä tulee tällöin joko tiedustelija, joka alkaa tutkia ympäristöään ravinnonlähteitä etsien, tai seuraaja, seuraten tällöin mehiläistanssia ja värväytyä tämän ravinnonlähteen hakuun. Kun mehiläinen löytää ravinnonlähteen,

se säilöö tähän sisältyvän informaation ja hyödyntää ravinnonlähdettä. Ravinnonetsijä palaa tämän jälkeen pesälle ja valitsee seuraavan toimintonsa kolmesta toimintavaihtoehdosta:

1. Mehiläinen saattaa hylätä pesän, jolloin mehiläisestä tulee passiivinen ravinnonetsijä.

2. Mehiläinen saattaa suorittaa mehiläistanssin, värväten parven muita jäseniä tälle ravinnonlähteelle.

3. Mehiläinen saattaa jatkaa löytämänsä ravinnonlähteen hyödyntämistä, mutta olla suorittamatta mehiläistanssia.

Näitä mekanismeja hyväksikäyttäen mehiläisyhdyskunta saavuttaa ravinnonetsinnässään ominaisuudet, joihin perustuu parven parviälykkyysmäinen itseorganisointi: positiivinen ja negatiivinen palaute, satunnaisvaihtelut toiminnassa ja populaatioperustainen monen toimijan keskinäinen interaktio. [Karaboga, 2005]

Mehiläisyhdyskuntaoptimointialgoritmin toteutuksessa puolet käytössä olevasta yhdyskunnasta määritellään seuraajiksi ja puolet aktiivisiksi mehiläisiksi. Jokaista aktiivista mehiläistä kohden on yksi ravinnonlähde pesän ympäristössä. Menetelmää suppeasti kuvaava korkean tason pseudokoodi on esitelty algoritmissa 5. [Karaboga and Basturk, 2007]

Algoritmi Mehiläisyhdyskuntaoptimointi
Tuloste: Paras löydetty ratkaisu $\boldsymbol{x}_*$
    $Alusta()$
    while $not\ lopetusehto\_voimassa()$ **do**
        $TyöllistetyMehiläisetRavinnonlähteille()$
        $SeuraajaMehiläisetRavinnonlähteille()$
        $LähetäTiedustelijat()$
    end while
    $return\ \boldsymbol{x}_*$
end Mehiläisyhdyskuntaoptimointi

Algoritmi 5: Mehiläisyhdyskuntaoptimointialgoritmin korkean tason pseudokoodi.

Algoritmin 5 alustusvaiheessa ravinnonlähteet valitaan satunnaisesti ja valittujen ravinnonlähteiden laatu määritetään. Laadunmäärityksen jälkeen mehiläiset palaavat pesälle ja jakavat ravinnonlähteitä koskevan informaation parven kanssa. Jaettuaan informaation, aktiiviset mehiläiset siirtyvät ravinnonlähteillä, jotka näillä on muistissa ja valitsevat uuden ravinnonlähteen ympäristöään tarkkailemalla. Seuraajamehiläiset valitsevat ravinnonlähteen aktiivisten mehiläisten suorittaman mehiläistanssin perusteella ja siirtyvät aktiivisten mehiläisten tapaan kohti ravinnonlähdettä. Algoritmissa korkeintaan yksi tiedustelija kerrallaan etsii uutta ravinnonlähdettä ja aktiivisten ravinnonetsijöiden ja seuraajien lukumäärä on sama. [Karaboga and Basturk, 2007]

Ravinnonlähteiden sijainteja käsitellään mehiläisyhdyskuntaoptimointialgoritmissa optimointiongelman ratkaisukandidaatteina. Kuten todettua, mehiläisryhmien koot vastaavat ravinnonlähteiden määrää ja täten myös ratkaisukandidaattien määrää. Alustuksessa generoidaan satunnaiset ratkaisukandidaatit populaationa $P(G = 0)$, joka koostuu ratkaisukandidaattien joukosta kooltaan SN. G merkitsee populaation sukupolvea. Ravinnonlähteitä merkitään D-ulottuvuuksisilla vektoreilla $x_i, i = 1, \dots, SN$, ratkaisukandidaattivektoreissa käytettävän ulottuvuuksien määrän määräytyessä käsiteltävän optimointiongelman parametrimäärän mukaisesti. Alustusvaiheen jälkeen algoritmin mehiläisprosesseja iteroidaan ratkaisuja $\boldsymbol{x_i} \in P$ vasten, kunnes lopetusehto täyttyy. Lopetusehto voi olla esimerkiksi iteraatiomaksimin C_{max} saavuttaminen. Suorituksen aikana mehiläiset muokkaavat stokastisesti ratkaisupopulaatiota mukaillessaan uusien ravinnonlähteiden löytämistä, vertaillen tässä yhteydessä ratkaisukandidaattien laatua suorittamalla tavoitefunktiota ratkaisukandidaatteja vasten. Mikäli uusi ratkaisukandidaatti on laadullisesti aiemmin tunnettua parempi, korvaa mehiläinen muistissaan aiemmin säilötyn ratkaisun löytyneellä ratkaisulla. Seuraajamehiläiset toimivat samoin, mikäli ne tulevat värvätyksi johonkin ravinnonlähteeseen. Seuraaja valitsee ravinnonlähteen stokastisesti mehiläistanssissa todennäköisyydellä

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n}, \tag{28}$$

missä p_i on seuraajan todennäköisyys valita ratkaisukandidaatti i ja fit_n on aktiivisen mehiläisen arvioima ratkaisukandidaatin i sopivuusarvo, joka on verrannollinen sijainnin i ravinnonlähteen arvoon. Uuden ratkaisun tuottamisessa mehiläisyhdyskuntaoptimointialgoritmissa sovelletaan kaavaa

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), \tag{29}$$

missä v_{ij} on uusi ratkaisukomponentti, $j = \{1, \dots, D\}$ ja $k = \{1, \dots, BN\}, k \neq i$ valitaan satunnaisesti. $BN = SN$ on aktiivisten mehiläisten kokonaismäärä ja ϕ_{ij} on satunnaisluku lukuväliltä $[-1,1]$. Satunnaislukuparametri ϕ_{ij} ohjaa uuden ratkaisukandidaatin generointia tunnetun ravinnonlähteen x_{ij} ympärillä. On huomionarvoista, että kun erotus $x_{ij} - x_{kj}$ pienenee, vähenee myös ratkaisukandidaatin muutoksen vaihtelu, mistä seuraa haun tehostamisvaikutus, kun haun askelmitta kutistuu adaptiivisesti seuraten haun lähenemistä kohti hakuavaruuden optimeita. [Karaboga and Basturk, 2007]

Algoritmissa voidaan käyttää rajoitusparametreja rajoittamaan ratkaisukandidaatin sijainnin päivitys halutulla arvoalueelle. Mikäli mehiläisen muistiin tallennetun ratkaisun laatua ei voida enää parantaa esimääritellyn iteraatiomäärän kuluessa, se hylätään. Ravinnonlähteen hylkäys toteutetaan mehiläisyhdyskuntaoptimointialgoritmissa generoimalla satunnaisesti uusi sijainti ja korvaamalla hylätty ratkaisukandidaatti tällä uudella sijainnilla. Kun mehiläisen jokainen kandidaattiratkaisu v_{ij}

on rakennettu, verrataan sen laatua tunnettuun ratkaisuun x_{ij}. Laadullisesti parempi ratkaisukandidaatti säilytetään mehiläisen muistissa. [Karaboga and Basturk, 2007]

Karaboga ja Basturk [2007] esittävät algoritmissa suoritettavan neljänlaista valintamenettelyä:

1. Globaali valinta, jota suoritetaan seuraajamehiläisten valitessa lupaavia alueita hakuavaruudesta.

2. Paikallinen valinta, jota suoritetaan mehiläisten valitessa uutta ratkaisukandidaattia tuntemansa ratkaisun ympäriltä.

3. Ahne paikallinen valinta, jota suoritetaan mehiläisten valitessa muistiinsa tallennettavaa vaihtoehtoa ratkaisukandidaateista näiden laadullisten ominaisuuksien perusteella.

4. Satunnainen valinta, jota suorittavat tiedustelijamehiläiset etsiessään uusia ratkaisukandidaatteja.

Mehiläisyhdyskuntaoptimointialgoritmin suoritusta voidaan ohjata kolmella säätöparametrilla: SN, C_{max} ja $limit$. Parametri $limit$ kuvaa iteraatiomäärää, jonka kuluttua ratkaisukandidaatti hylätään, mikäli tämä ei ole parantunut.

Mehiläisyhdyskuntaoptimointialgoritmin julkaisun yhteydessä Karaboga ja Basturk [2007] testasivat menettelyä muutamia tunnettuja multimodaalisia numeerisen optimoinnin testifunktioita vasten. Mukana vertailussa olivat myös geneettiset algoritmit, partikkeliparvioptimointi sekä partikkeliparvioptimoinnin ja geneettisen algoritmin hybridi, PS-EA (Particle Swarm Inspired Evolutionary Algorithm). Vertailussa käytettiin mehiläisyhdyskuntaoptimoinnin parvipopulaationa lukumäärää 125 ja iteraatiomaksimi riippui käytettävästä ulottuvuusmäärästä siten, että ulottuvuusmääriä 10, 20 ja 30 vastasivat iteraatiomaksimit 500, 750 ja 1000. Vertailussa havaittiin, että menettely on hyvin kilpailukykyinen suhteessa verrokkeihinsa. Se jäi suorituskyvyssä toiseksi geneettiselle algoritmille ja verratulle hybridialgoritmille murto-osassa vertailuita, mutta osoitti pääsääntöisesti parempaa suorituskykyä. Erityishuomiona on todettava mehiläisyhdyskuntaoptimoinnin kyky hakeutua pois lokaaleista optimeista globaalia optimia tavoitellessa. Algoritmin todettiin vertailun perusteella olevan kilpailukykyinen multimodaalisten optimointiongelmien ratkaisemisessa.

Civicioglu ja Besdok [2011] ovat vertailleet koeasetelmissaan tilastollisia menetelmiä käyttäen eri optimointiheuristiikkojen suorituskykyä. Vertailuissa oli mehiläisyhdyskuntaoptimointi-algoritmin ohella mukana partikkeliparvioptimointi, *käkihaku* ja Stornin ja Pricen [1997] kehittämä *DE -metaheuristiikka* (differential evolution). Vertailuryhmästä mehiläisyhdyskuntaoptimointi suoriutui kokonaisuutena heikoimmin. Sen kuitenkin todettiin käsittävän menestyksekkäitä strategioita algoritmissa tehtävään päätöksenteon ja ratkaisukandidaattien käsittelyyn liittyen. Optimointimetaheuristiikkana se sisältää siis lupaavia osa-alueita ja saattaa parantaa kilpailukykyään entisestään jatkokehittelyllä ja säätöparametrien oikeellisella valinnalla.

4.4. Harmaasusioptimoija

Harmaasusioptimoija (Grey Wolf Optimizer) on Mirjalilin ja muiden [2014] kehittämä optimointimetaheuristiikka, joka perustuu harmaasusien laumakäyttäytymiseen metsästyksen yhteydessä. Harmaasusilaumat ovat luonnossa järjestäytyneet tiukkaan hierarkiaan: laumaa johtaa uros-naaras-pari, joita kutsutaan *alfoiksi*. Alfat tekevät päätökset koskien lauman metsästystä ja muuta toimintaa. Alfan alapuolella susilauman hierarkiassa ovat *beetat.* Beeta-susi voi olla koiras tai naaras ja tämän toimenkuvana on avustaa alfa-sutta lauman johtamisessa. Beeta-susi on myös usein todennäköisin vaihtoehto korvaamaan alfan, mikäli tämä menettää roolinsa. Alimpana susien hierarkiassa ovat *omegat.* Omegan toimena on alistua lauman muille susille ja purkaa lauman sisäisiä jännitteitä sekä muiden jäsenten turhaumia olemalla kurinpidon ja dominanssin osoitusten kohteena. Lauman susi, joka ei kuulu yllä määriteltyihin ryhmiin, on alamainen tai *delta*, joka kuuluu laumahierarkiassa beetan ja omegan väliin. [Mirjalili et al., 2014]

Sudet metsästävät laumassa. Lauman metsästyskäyttäytyminen jakautuu Muron ja muiden [2011] mukaan kolmeen vaiheeseen:

1. saaliin lähestyminen ja jäljitys,
2. saaliin jahtaaminen, häirintä ja piiritys, ja
3. hyökkäys.

Harmaasusioptimoijassa mallinnetaan susilauman sosiaalinen hierarkia siten, että määritellään keskenään eriarvoiset ratkaisukandidaatit susilauman jäsenyyskategorioiden mukaisesti. Parasta ratkaisua merkitään alfalla (α), toiseksi parasta beetalla (β) ja kolmanneksi parasta deltalla (δ). Muita ratkaisuja merkitään omegalla (ω). Ratkaisun hakeminen harmaasusioptimoijassa toteutetaan käyttäen mallina susilauman piirityskäyttäytymistä, mallintamalla tämä käyttäen vektoreita. Piirityskäyttäytymisen perustana Mirjalili ja muut [2014] esittelevät kaavat

$$\vec{D} = \left|\vec{C} \cdot \vec{\mathrm{X}}_p(t) - \vec{X}(t)\right|, \tag{30}$$

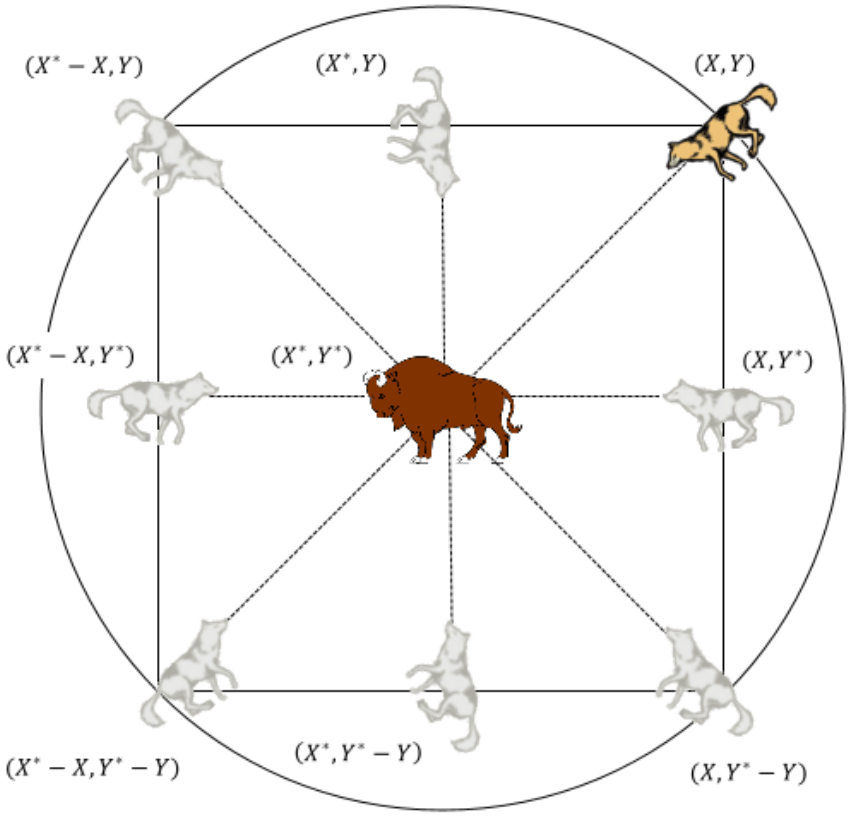
$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D}, \tag{31}$$

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a} \text{ ja} \tag{32}$$

$$\vec{C} = 2\vec{r}_2, \tag{33}$$

missä $\vec{X}_p$ merkitsee saaliin sijaintia, $\vec{X}$ suden sijaintia, t nykyistä iteraatiota ja $\vec{A}$, $\vec{C}$, $\vec{a}$, $\vec{r}_1$ sekä $\vec{r}_2$ ovat kerroinvektoreita, missä vektorin $\vec{a}$ komponentteja vähennetään lineaarisesti arvosta 2 arvoon 0 ja $\vec{r}_1$ ja $\vec{r}_2$ ovat satunnaisvektoreita välillä $[0, 1]$. Kaavoista (30), (31), (32) ja (33) seuraa, että kun sutta merkitsevällä ratkaisua etsivällä agentilla on tiedossaan jokin saalista edustava ratkaisukandidaatti, voi suden sijainti päivittyä mihin tahansa sijaintiin havaitun ratkaisukandidaatin ympärillä. Kuva 4

havainnollistaa piiritystilannetta, jossa X^*, Y^* ja X, Y edustavat vastaavasti saaliin ja suden sijaintia ajanhetkellä t.



Kuva 4. Suden piirityskäyttäytyminen harmaasusioptimoijassa [Mirjalili et al., 2014].

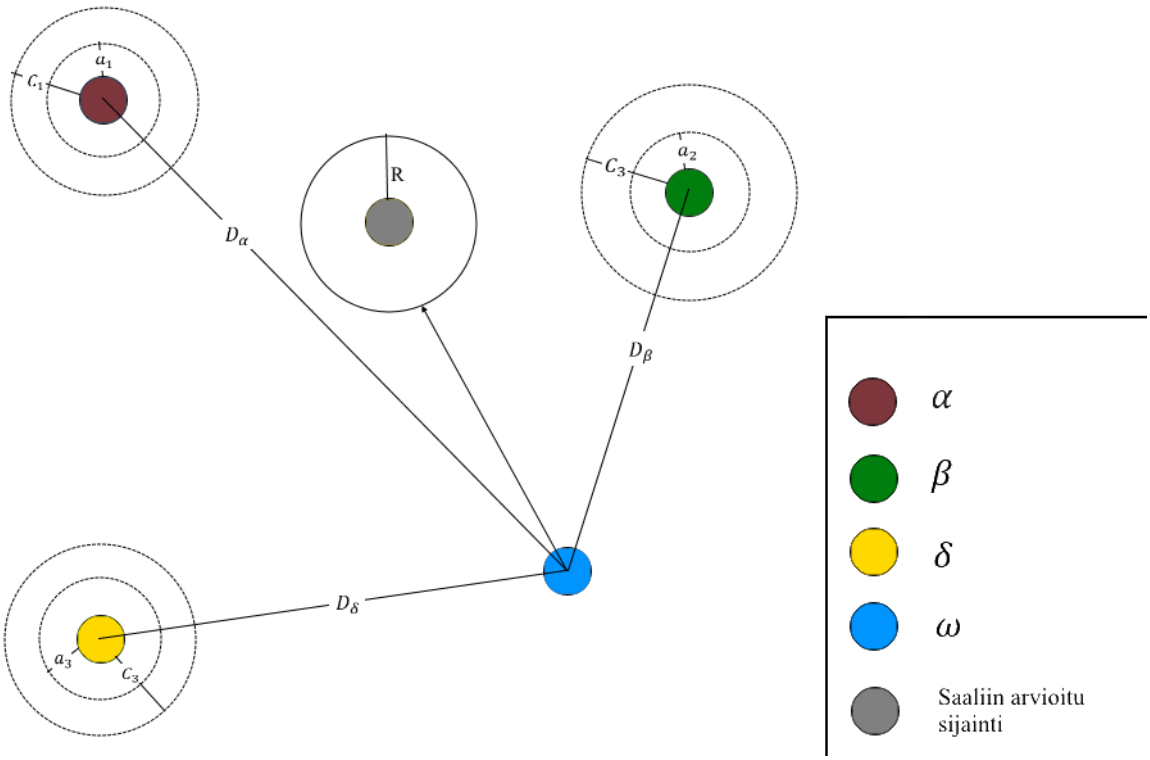
Sosiaalilinen hierarkia vaikuttaa susien metsästykseen siten, metsästyksen kulkua ohjaa useimmiten alfa, joskin myös beetat ja deltat voivat osallistua metsästykseen. Muut sudet seuraavat metsästystä ohjaavia susia. Harmaasusioptimoijassa tätä käyttäytymistä mallinnetaan siten, että oletetaan alfa-, beeta- ja delta-agenttien edustavan parasta informaatiota ratkaisun sijainnista. Alfa edustaa menetelmässä parasta tunnettua ratkaisua. Käyttäytymistä, jossa muut sudet seuraavat metsästystä johtavia susia saaliin etsimisessä, mallinnetaan harmaasusioptimoijassa tallentamalla kolme ensimmäisenä löydettyä parasta ratkaisukandidaattia ja päivittämällä muiden susien sijaintia näiden pohjalta. Mirjalili ja muut [2014] ehdottavat tämän toteuttamiseksi kaavat

$$\vec{D}_\alpha = \left|\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}\right|, \vec{D}_\beta = \left|\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}\right|, \vec{D}_\delta = \left|\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}\right|, \tag{34}$$

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot \left(\vec{D}_\alpha\right), \vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot \left(\vec{D}_\beta\right), \vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot \left(\vec{D}_\delta\right), \tag{35}$$

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}, \tag{36}$$

missä $\vec{X}_\alpha$, $\vec{X}_\beta$ ja $\vec{X}_\delta$ ovat vastaavasti alfa-, beeta- ja delta-agenttien sijainnit hakuavaruudessa. Kaavojen sovellutuksesta seuraa, että haku kohdistuu alueelle, jonka määräävät alfa-, beeta- ja delta-agenttien määrittelemät alueet yhdessä. Tämä merkitsee sitä, että alfa-, beeta- ja delta-agentit tekevät arvioita ratkaisukandidaatin sijainnista ja muut agentit päivittävät sijaintiaan satunnaisesti ratkaisukandidaatin ympärillä mukaillen saalistaan piirittäviä susia. Kuva 5 havainnollistaa agenttien sijainnin päivitystä kaavan (36) mukaisesti [Mirjalili et al., 2014].



Kuva 5. Harmaasusioptimoijan hakuagenttien sijaintien päivitys [Mirjalili et al., 2014].

Saalista kohti lähestyminen tapahtuu menettelyssä vähentämällä kaavan (32) kerroinvektorin $\vec{a}$ komponenttien arvoja vähitellen kohti nollaa, mistä seuraa kerroinvektorin $\vec{A}$ arvoalueen $[-2a, 2a]$ supistuminen. Kun vektorin $\vec{A}$ komponentit ovat väliltä $[-1, 1]$, pakottaa tämä agentin lähestymään ratkaisua, mukaillen susien hyökkäämistä. Kun taas komponentit ovat suurempia kuin 1 tai pienempiä kuin -2, ohjaa tämä agentin poispäin ratkaisusta etsien uutta vaihtoehtoa. [Mirjalili et al., 2014]

Satunnaisuutta hakuun tuovilla elementeillä pyritään välttämään lokaaliin optimiin jäämistä korostamalla hakuavaruuden tutkimista. Harmaasusioptimoijassa satunnaiskerroinvektorin $\vec{A}$ komponenttivektorin $\vec{a}$ komponenttien arvojen pienentämisestä seuraa hakuavaruuden tutkimisen painoarvon vähentyminen suhteessa parhaan tunnetun arvon lähentymiseen. Tämän sijaan satunnaiskerroinvektorin $\vec{C}$ arvoalue tai painotus ei muutu iteraatioiden edetessä, vaan tämä saa algoritmin suorituksen alusta loppuun satunnaisia arvoja väliltä $[0, 2]$. Sen tarkoituksena on korostaa

tai vähentää ratkaisun sijainnin merkitystä mallinnettaessa etäisyyttä suhteessa agentin sijaintiin. Mikäli vektorin $\vec{C}$ saamat arvot ovat suurempia kuin yksi, antaa tämä ratkaisun sijainnille ylimääräisen painotuksen, ja vastaavasti arvoilla, jotka ovat pienempiä kuin yksi, vähenee ratkaisun sijainnin merkitys määriteltäessä vektoria $\vec{D}$. Suhteessa analogiaan harmaasusien metsästyksestä vektorilla $\vec{C}$ pyritään simuloimaan luonnossa esiintyviä esteitä suden ja tämän saaliin välillä. Metaheuristiikassa tällä vältetään lokaaliin optimiin jäämistä korostamalla etsinnän satunnaiselementtiä ja mahdollistamalla hakuavaruuden tutkiminen myös algoritmin myöhemmillä iteraatioilla. [Mirjalili et al., 2014]

Pseudokoodi harmaasusioptimoijasta on kuvattu algoritmissa 6.

Algoritmi Harmaasusioptimoija
Syötteet: Harmaasusipopulaatio $X_i, (i = 1, 2, \dots, n)$, tavoitefunktio $f(x)$, iteraatiomaksimi max
Tuloste: Paras löydetty ratkaisu X_α

$Alusta(a)$
$Alusta(A)$
$Alusta(C)$
$F_i = f(X_i)$
$X_\alpha = paras(F_i)$
$X_\beta = toiseksi\_paras(F_i)$
$X_\delta = kolmanneksi\_paras(F_i)$
while $t \leq max$ **do**
  while $i \leq n$ **do**
    $Päivitä(X_i)$ // Kaavan (35) mukaisesti
    $i = i + 1$
  end
  $Päivitä(a)$
  $Päivitä(A)$
  $Päivitä(C)$
  $F_i = f(X_i)$
  $X_\alpha = paras(F_i)$
  $X_\beta = toiseksi\_paras(F_i)$
  $X_\delta = kolmanneksi\_paras(F_i)$
  $t = t + 1$
end while
return X_α
end Harmaasusioptimoija

Algoritmi 6: Harmaasusioptimoija-algoritmin pseudokoodi.

Mirjalili ja muut [2014] ovat testanneet harmaasusioptimoijaa 39:llä yleisesti optimointimenetelmien tutkimuksessa käytetyllä funktiolla ja osoittaneet harmaasusioptimoijan saavuttavan testeissä hyviä tuloksia muihin tunnettuihin optimointiheuristiikkoihin verrattuna. Verrokkimenetelminä toimivat partikkeliparvioptimointi, gravitaatiohakualgoritmi sekä kolme evolutiivista algoritmia: differentiaalinen evoluutio, nopea evolutiivinen ohjelmointi sekä evoluutiostrategia kovarianssimatriisisovellutuksella. Osa testeissä käytetyistä funktioista on ns. *komposiittifunktioita* (composite functions), joiden avulla voidaan testata samanaikaisesti optimointimenetelmän kykyä tutkia hakuavaruutta sekä lähentyä ratkaisukandidaatteja. Tällaiset funktiot sisältävät myös suuren määrän lokaaleita optimeita. Näissä testeissä harmaasusioptimoija suoriutui verrokkimenetelmien joukosta parhaiten, mikä osoittaa menetelmältä hyvää tasapainoa hakuavaruuden kartoituksen, sen hyödyntämisen sekä lokaalien optimien välttämäisen kanssa. [Mirjalili et al., 2014]

Bergh ja Engelbrecht [2006] ovat todenneet, että hakuavaruuden kartoittamiseksi parviälykkyysalgoritmissa tulisi hakuagenttien sijainneissa tapahtua suorituksen alkuvaiheessa äkillisiä liikkeitä. Algoritmin suorituksen edetessä sijainnin vaihteluiden tulisi vähentyä, jotta hakutuloksia voidaan hyödyntää ja agentit voivat lähentyä optimia. Harmaasusioptimoijan on osoitettu käyttäytyvän näiden ehtojen mukaisesti, ensin korostaen suorituksensa alussa hakuavaruuden tutkimista ja kerroinvektorin $\vec{a}$ arvojen pienentyessä hakutulosten hyväksikäyttöä hakuagenttien liikkeen painottuessa kohti löydettyjä ratkaisukandidaatteja [Mirjalili et al., 2014]. Tällainen käyttäytyminen hakuagenttien muodostaman parven liikkeissä takaa parven etsinnän konvergoituvan kohti jotain hakuavaruuden pistettä [Berg and Engelbrecht, 2014].

4.5. Lepakkoalgoritmi

Lepakkoalgoritmi (Bat algorithm) on verrattain uusi lisäys parviälykkyysmetaheuristiikkojen joukkoon. Sen kehitti Yang, [2010] tavoitteenaan yhdistellä hyviä puolia aiemmin tunnetuista optimointiheuristiikoista minimoiden menetelmän puutteet siten, että saataisiin muodostettua tunnettuja menetelmiä parempi algoritmi. Lepakkoalgoritmin innoituksena on monilla lepakkolajeilla luonnossa ilmenevä kaikuluotaukseen perustuva navigointi. Kaikuluotausta hyväksikäyttäen lepakot kykenevät paikantamaan saaliinsa usein ilman tarvetta muille aistihavainnoille.

Useimmat lepakkolajit hyödyntävät kaikuluotausta jollain tavalla. Ne käyttävät ravinnokseen hyönteisiä ja metsästyksessään ne käyttävät kaikuluotausta hyönteisten paikantamiseen ja yleiseen navigointiin. Kaikuluotauksessa lepakko päästää kovan äänen ja pinnoista kimpoavan äänikaiun perusteella muodostaa kuvan ympäristöstään. Äänen luonne saattaa vaihdella tilanteen tai lepakkolajin mukaan eri lajien käyttäessä hyväkseen ääniä eri taajuuksilla. Yleensä lepakon päästämän äänen taajuus on välillä $24\,kHz - 100\,kHz$ ja yksittäisen ääni-impulssin kesto $5 -$

$20\ ms$. Lepakot voivat parhaimmillaan päästää tällaisia ääni-impulsseja jopa 200 kertaa sekunnissa. Äänen aallonpituus saadaan kaavasta

$$\lambda = \frac{v}{f}, \tag{37}$$

missä λ on aallonpituus, $v = 340\ m/s$ on äänen nopeus ja f on vakiollinen äänen taajuus. Taajuuksille $25\ kHz - 150\ kHz$ vastaava aallonpituusväli on $2\ mm - 14\ mm$, joka vastaa lepakoiden metsästämien saaliseläinten kokoluokkaa. Pulssien äänenvoimakkuus voi olla korkeimmillaan $110\ dB$ ja täten äänen kantama joitakin metrejä. Kaikuluotauspalautteen tulkinnassa ympäristömallin muodostamiseksi lepakot hyödyntävät useita eri menetelmiä:

1. Aikaviivettä äänen päästämisen ja kaiun vastaanottamisen välillä,
2. korvien keskinäisen ääniaistimuksen välistä aikaeroa, ja
3. kaikuaistimusten välisiä äänenvoimakkuusarvoja.

Näillä keinoin lepakot pystyvät tunnistamaan ympäristöstään saaliin, sen tyypin, etäisyyden, suunnan ja liikkumisnopeuden. [Yang, 2010]

Lepakkoalgoritmin pohjalla käytettävät mekanismit mallinnetaan perustuen tähän lepakkojen metsästyksessään ja navigoinnissaan käyttämään kaikuluotausperustaiseen mekanismiin. Mallinnuksessa pohjataan yksinkertaistettuihin taustaoletuksiin kaikuluotauksesta:

1. Kaikki lepakot aistivat etäisyyttä kaikuluotauksella ja kykenevät automaattisesti erottamaan saaliin muista objekteista.
2. Lepakot liikkuvat lentäen ympäristössään satunnaisesti seuraavin ominaisuuksin: nopeus $\boldsymbol{v}_i$, sijainti $\boldsymbol{x}_i$, äänen taajuus f_{min}, sen aallonpituus λ ja voimakkuus A_o. Ne päästävät ääniä kiinteällä taajuudella ja voivat säätää päästämänsä äänen aallonpituutta sekä äänen päästämisen tiheyttä suhteessa saaliinsa etäisyyteen. Äänen päästämisen tiheyttä ilmaistaan parametrilla $r \in [0,1]$.
3. Äänenvoimakkuus vaihtelee välillä $[A_{mim}, A_0]$. Äänen taajuus vaihtelee välillä $[0, f_{max}]$.
4. Laskennallisen tehokkuuden säilyttämiseksi aikaviivearvioita ei oteta mallissa huomioon. Tätä voitaisiin tehdä säteenseurantaa hyväksikäyttäen, mutta on laskennallisesti vaativaa useiden ulottuvuuksien tapauksissa.
5. Oletetaan, että taajuusaluetta $[f_{min}, f_{max}]$ vastaa tietty aallonpituuksien arvoalue $[\lambda_{min}, \lambda_{max}]$. Esimerkiksi taajuusaluetta $[20\ kHz - 500\ kHz]$ vastaa aallonpituuksien alue $[0.7\ mm - 17\ mm]$.

Aallonpituudet toimivat mallinnuksessa säätöparametreina, joita säädetään kulloisenkin sovelluskohteena olevan ongelman asettamien vaatimusten mukaisesti. Vaihtoehtoisesti aallonpituuden sijaan voidaan kiinnittää aallonpituus ja käyttää säätöparametrina taajuutta, sillä λf on vakiollinen. [Yang, 2010]

Lepakkojen liikettä mallinnetaan algoritmissa hieman partikkeliparvioptimoinnin kaltaisesti vektoreina. Lepakoiden käyttämän taajuuden, niiden liikenopeuden ja sijainnin päivityksissä hyödynnetään kaavoja

$$f_i = f_{min} + (f_{max} - f_{min})\beta, \tag{38}$$

$$\boldsymbol{v}_i^t = \boldsymbol{v}_i^{t-1} + (\boldsymbol{x}_i^t - \boldsymbol{x}_*)f_i, \tag{39}$$

$$\boldsymbol{x}_i^t = \boldsymbol{x}_i^{t-1} + \boldsymbol{v}_i^t, \tag{40}$$

missä $\boldsymbol{x}_i^t$, f_i ja $\boldsymbol{v}_i^t$ ilmaisevat vastaavasti lepakon i sijaintia, taajuutta ja liikenopeutta ajanhetkellä t, $\boldsymbol{x}_*$ on paras tähän mennessä löydetty ratkaisu ja β on satunnaismuuttuja tasaisesti jakautuneelta lukuväliltä [0,1]. Toisin sanoen kaavassa (38) valitaan käytettävä taajuus f_i lukuvälin $[f_{min}, f_{max}]$ määrittelemästä tasajakaumasta. Yangin [2010] alkuperäisessä implementaatiossa tämä lukuväli määritellään väliksi $[0, 100]$. Näiden liikkeiden ohella lepakoille generoidaan paikallista satunnaiskulkua hyväksikäyttäen uudet sijainnit kaavalla

$$\boldsymbol{x}_{new} = \boldsymbol{x}_{old} + \epsilon A^t, \tag{41}$$

missä ϵ on satunnaisluku väliltä $[-1, 1]$ ja A^t on koko lepakkopopulaatiosta laskettu keskimääräinen äänenvoimakkuus ajanhetkellä t. Satunnaiskulun tavoitteena on etsinnän tehostaminen. [Yang, 2010]

Tehostamisvaikutuksen aikaansaamiseksi on algoritmin suorituksen edetessä säädettävä käytettäviä parametreja. Sääntelyssä tavataan vähentää äänenvoimakkuutta ilmaisevaa parametria A ja kasvattaa vastaavasti äänen päästötiheyttä ilmaisevaa parametria r kun lepakko löytää saaliinsa. Parametrien päivitykset toteutetaan vastaavasti kaavoilla

$$A_i^{t+1} = \alpha A_i^t, \tag{42}$$

$$r_i^{t+1} = r_i^0[1 - \exp(-\gamma t)], \tag{43}$$

missä α ja γ ovat päivitystä ohjaavia vakioparametreja. Näille parametreille pätee seuraavat ominaisuudet:

$$A_i^t \to 0, kun\ t \to \infty\ , a \in]0,1[, \tag{44}$$

ja

$$r_i^t \to 0, kun\ t \to \infty, \gamma > 0. \tag{45}$$

Yangin [2010] alkuperäisessä implementaatiossa näiden parametrien arvoiksi on valittu $\alpha = \gamma = 0.9$. Alkuarvot äänenvoimakkuus- ja impulssitiheysparametreille tavataan jakaa lepakkopopulaatiolle satunnaistetusti joltain lukuväliltä, esimerkiksi $A_i^0 \in [1,2]$. Parametreja päivitetään suorituksen edetessä aina, kun etsinnässä saavutetaan parempia tuloksia, mistä seuraa haun tehostamisvaikutusta. On huomionarvoista, ettei ole ensiarvoisen tärkeää, kuinka suuret äänenvoimakkuusparametrien saamat arvoalueet ovat, kunhan noudetaan sääntöä, jossa äänenvoimakkuus laskee impulssitiheyden kasvaessa, kun saalista lähestytään – alue $[A_{min}, A_0]$ voi olla esimerkiksi $[0,1]$ tai $[0,100]$. [Yang, 2010]

Pseudokoodi lepakkoalgoritmista on kuvattu algoritmissa 7.

Algoritmi Lepakkoalgoritmi
Syötteet: Lepakkopopulaatio $\boldsymbol{x}_i, (i = 1, 2, \dots, n)$, tavoitefunktio $f(x)$, iteraatioiden maksimimäärä max,
Tuloste: Paras löydetty ratkaisu $\boldsymbol{x}_*$

```
Alusta(x_i, v_i, f_i)
Alusta(r_i, A_i)
while t ≤ max do
    Generoi ratkaisut säätämällä taajuutta ja päivittämällä x_i
    if rand > r_i do
        SatunnaisKulku(x_*)
    end if
    EtsiRatkaisut()
    if rand < A_i && f(x_i) < f(x_*)
        HyväksyRatkaisut()
        Kasvata(r_i)
        Vähennä(A_i)
    end if
    x_* = Paras(x_i)
end while
```

end Lepakkoalgoritmi

Algoritmi 7: Lepakkoalgoritmin pseudokoodi.

Lepakkoalgoritmin suorituskykyä on testattu kokeellisesti käyttäen tavoitefunktioina muutamaa tunnettua jatkuvan optimoinnin mittausfunktiota, muun muassa Rosenbrockin funktiota

$$f(x) = \sum_{i=1}^{d-1} (1 - x_i^2)^2 + 100(x_{i+1} - x_i^2)^2, -2048 \leq x_i \leq 2.048. \quad (46)$$

Vertailualgoritmeina koeasettelussa oli standardimuotoinen geneettinen algoritmi sekä partikkeliparvioptimointi-implementaatio. Lepakkoalgoritmin säätöparametrina käytettiin $\alpha = 0.9$ ja lepakkopopulaation koko n oli välillä $25 - 50$. Vertailussa havaittiin, että partikkeliparvioptimointimenetelmä suoriutui huomattavasti geneettistä algoritmia paremmin ja lepakkoalgoritmi puolestaan huomattavasti paremmin kuin partikkeliparvioptimointi sekä suorituskyvyn tehokkuuden, että löydettyjen ratkaisuiden laadun osalta. [Yang, 2010]

Lepakkoalgoritmin jatkeeksi on tämän varianttina kehitetty myös algoritmi monen muuttujan optimointiin. Koeasetelmissa on käytetty tunnettuja monen muuttujan optimointialgoritmien suorituskyvyn mittaamiseen tarkoitettuja testifunktioita ja saatujen tulosten perusteella voidaan todeta, että tämäkin lepakkoalgoritmin variantti on verrokkeihinsa nähden kilpailukykyinen ratkaisumenetelmä ja lupaava kandidaatti jatkotutkimuksia varten. [Yang, 2011]

4.6. Käkihaku

Käkihaku (Cuckoo search) on tuoreehko parviälykkyysmetaheuristiikka, jonka ovat kehittäneet Yang ja Deb [2009], pääasiallisena innoituksenaan joillekin käkilajeille ominainen, näiden selviytymiselle välttämätön pesäloisintakäyttäytyminen. Toisena luonnosta valittuna innoitteena algoritmissa toimii joidenkin lintu- ja hyönteislajien liikkumiselle ominainen *Lévy-kävely* (Lévy flight), joka kuvaa näiden lajien taipumusta liikkua kerrallaan pitkiä yksittäisiä liikkeitä useiden, lyhempien ja satunnaisempien liikkeiden jaksottelemana.

Vaikka kaikkien käkilajien pesimäkäyttäytymiseen piirre ei kuulukaan, harrastavat useat käkilajit lisääntymisessään pesäloisintaan. Pesäloisinnassa käki munii munansa toisten lintujen pesiin, jolloin resursseja säästyy hautomis- ja hoivakäyttäytymiseltä. Käki munii munansa samoihin aikoihin kuin pesän omistaja, ja käet kuoriutuvat näin usein pesän muita munia aikaisemmin. Käenpoikaset poistavat muut kuoriutumattomat munat pesästä, jolloin niille riittää entistä suurempi osa pesän omistajan tarjoamasta hoivasta. Pesäloisintaa esiintyy luonnossa yleisesti useissa eri muodoissa: se voi olla lajin sisäistä loisintaa, yhteisöllistä pesintäkäyttäytymistä tai toisten lajien pesien haltuunottoa. Käkilajien pesintäkäyttäytymiseen lukeutuu näistä kahta jälkimmäistä tyyppiä. [Yang and Deb, 2009]

Lévy-kävely on ranskalaisen matemaatikon Paul Lévyn mukaan nimetty satunnaiskävelyiden luokka. Se määrittelee satunnaiskävelyitä, joiden askelpituudet poimitaan niin sanotusta Lévyn satunnaisjakaumasta, jonka ominaispiirteenä on, että jakauman hännät ovat paksuja, eli eivät eksponentiaalisesti rajattuja. Useilla eläinlajeilla ja jopa joillain ihmisheimoilla ilmenee liikkumiskäyttäytymisessä piirteitä, jotka mukailevat Lévyn kävelyitä. Lévy-kävelyitä on perinteisesti käytetty fysiikan tutkimuksessa mallintamaan erilaisia prosesseja, kuten diffuusiota. Kävelyiden on myös osoitettu olevan erinomaisesti toimivia etsintämalleja tilanteissa, joissa etsitään harvasti ja satunnaisesti sijoittuneita kohteita. [Brown et al., 2007]

Mallinnettaessa käen pesimäkäyttäytymistä optimointiheuristiikassa, tehdään käyttäytymismalleista yksinkertaistavia taustaoletuksia:

1. Jokainen käkipopulaation jäsen munii kerrallaan yhden munan ja valitsee pesän, johon muninta tapahtuu, satunnaisesti.

2. Parhaat pesät jatkavat seuraaville sukupolville.

3. Mahdollisten pesien lukumäärä on vakio.

4. Pesän haltija tunnistaa munan tunkeutujaksi todennäköisyydellä $p_a \in [0,1]$, jolloin tämä joko poistaa munan pesästä tai hylkää pesän ja rakentaa uuden.

Mallinnuksessa optimointiheuristiikaksi voidaan pesiin sijoitetut munat mieltää ratkaisukandidaateiksi ja käen munat uusiksi ratkaisuiksi, joilla pyritään tarjoamaan korvaavia, parempia vaihtoehtoja valmiiksi tunnettuja ratkaisua edustaville pesässä valmiiksi oleville munille. [Yang and Deb, 2009]

Käkihaussa käytetään Lévy-kävelyitä ratkaisuiden generoinnin yhteydessä. Lévy-kävelyt, kuten satunnaiskävelyt yleensä, toimivat *Markovin ketjujen* mukaisella prosessilla: ne ovat stokastisia prosesseja, joissa prosessin seuraavan tilan määrittää ainoastaan sen tämänhetkinen tila ja seuraavan tilan siirtymätodennäköisyys. Käkihaun Levý-kävely suoritetaan kaavalla

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \oplus \text{Lévy}(\lambda), \tag{47}$$

missä $x_i^{(t)}$ on käen i generoima uusi ratkaisu ajanhetkellä t ja $\alpha > 0$ on ongelman kokoon suhteutettava askelmittaa kuvaava säätöparametri, yleensä $\alpha = 1$. Operaattori $\oplus$ merkitsee alkioittaista tuloa ja $Lévy(\lambda)$ parametrilla λ Lévy-jakaumasta poimittua satunnaismuuttujaa. Lévy-jakaumalla on ääretön varianssi sekä keskiarvo, jakauma kuvataan kaavalla

$$Lévy \sim u = t^{-\lambda}, (1 < \lambda \leq 3). \tag{48}$$

Lévy-kävely on satunnaiskävely, jossa käytettävä jakauma on potenssilain mukainen paksuhäntäinen todennäköisyysjakauma. Potenssilailla tarkoitetaan, että tiheysfunktion muuttujien välinen suhde on toisiinsa nähden eksponentiaalinen. Hyvien tulosten aikaansaamiseksi osa uusista ratkaisuista tulisi generoida parhaiden tunnettujen ratkaisuiden ympärille, mutta useimmat kuuluisi rakentaa satunnaisesti riittävän pitkillä etäisyyksillä parhaasta tunnetusta ratkaisusta, jotta hakumenettelyn hajauttamisvaikutus toimisi, eikä etsintä jumittuisi lokaaleihin optimeihin. Käkihaun pseudokoodi on esitelty algoritmissa 8. [Yang and Deb, 2009]

Käkihakualgoritmin referenssi-implementaatiota on testattu tunnettuja jatkuvan optimoinnin testausfunktioita vasten. Kokeissa haarukoitiin suorituskyvyn mittauksen ohella sopivia arvoja säätöparametreille, joiden osalta havaittiin, että useimpia optimointisovellutuksia ajatellen sopivat parametrien arvot ovat $n = 15$ ja $p_a = 0.25$, missä n on pesäpopulaation koko. Koeasetelmat osoittavat myös, etteivät säätöparametrien muokkaukset vaikuta dramaattisesti hakuprosessin konvergenssiin, mistä seuraten niiden sovellusongelmakohtainen hienosäätäminen ei ole tarpeellista.

Testifunktioiden joukossa oli muiden muassa kaavan (46) Rosenbrockin funktio. Kaikilla käytetyillä testifunktioilla käkihaun referenssi-implementaatio osoittautui paremmaksi kuin vertailualgoritmeina käytetyt geneettinen algoritmi sekä partikkeliparvioptimointi. Käkihaku oli jokaisella testifunktiolla sekä nopeampi ratkaisun löytämisessä että varmempi globaalin optimin löytämisessä. Syinä paremmuuteen voidaan esittää parempi tasapaino hajauttamisen ja tehostusvaikutuksen välillä sekä säätöparametrien vähäisempi määrä. [Yang and Deb, 2009]

Algoritmi Käkihaku
Syötteet: Isäntäpesien populaatio $\boldsymbol{x}_i, (i = 1, 2, \dots, n)$, tavoitefunktio $f(x)$, iteraatioiden maksimimäärä max,
Tuloste: Paras löydetty ratkaisu x_*

while $t \leq max$ **do**
$käki = Lévy()$
$F_i = f(käki)$
$F_j = PoimiPesäSatunnaisesti(n)$
if $F_i > F_j$ **do**
$j = i$
end if
$KorvaaPesätUusille(p_a)$
$SäilytäParhaat(\boldsymbol{x}_i)$
$Järjestä(\boldsymbol{x}_i)$
$x_* = EtsiParas(\boldsymbol{x}_i)$
end while
end Käkihaku

Algoritmi 8: Käkihakualgoritmin pseudokoodi.

Civicioglu ja Besdok [2011] ovat vertailleet tutkimuksessaan käkihakua muihin numeerisen optimoinnin menettelyihin, mukaan lukien partikkeliparvioptimointiin sekä ABC-algoritmiin. Vertailtavista algoritmeista parhaimmat tulokset saivat juuri käkihaku sekä alun perin Stornin ja Pricen [1997] kehittämä DE -metaheuristiikka (differential evolution). Testeissä korostui näiden menettelyjen vakaus suhteessa partikkeliparvioptimointiin, joka muuten on kilpailukykyinen ratkaisumenetelmä.

5. Sovellutuksia ja nykytila

Parviälykkyyden tutkimusala optimoinnissa on kasvanut valtaisasti alan syntyvaiheiden jälkeen, jolloin Marco Dorigon [1992] muurahaisyhdyskuntaoptimointi sekä Kennedyn ja Eberhartin [1995] partikkeliparvioptimointi julkaistiin. Näissä metaheuristiikoissa käytettyjä parviälykkyyden konsepteja on sovellettu eri alueilla kirjavasti, ja sekä muurahaisyhdyskuntaoptimointia, että partikkeliparvioptimointia on tutkimuksessa jatkokehitetty runsaasti. Marco Dorigo hallinnoi nykyään pelkästään parviälykkyysalgoritmeja koskevaa tutkimusta julkaisevaa tieteellistä lehtijulkaisua Swarm Intelligence. Uusi alaa käsittelevä tutkimus on johtanut mitä moninaisimpiin ja erikoistuneempiin muunnelmiin näistä kahdesta alkuperäisestä parviälykkyysoptimointialgoritmeista luettavasta tekniikasta. Tutkimusten joukosta löytyy muun muassa uudenlaisia variantteja vanhoihin parviälykkyysoptimointialgoritmeihin, [Omran et al., 2014] sekä näiden hybridisointeja ja tehostuksia muun muassa rinnakkaisuustekniikoilla [Abdelkafi et al., 2014]. Sovellutusten kirjavuudesta kertoo, että tekniikoita on hyödynnetty myös alueilla, jotka ovat kovin kaukana optimointialgoritmien käsittelyssä usein esiteltävien NP-täydellisten kombinatoristen optimointiongelmien sekä vaikeasti laskettavien numeeristen funktioiden alueelta: parviälykkyysoptimointialgoritmeja on sovellettu onnistuneesti muun muassa verkkosivujen käytettävyyteen liittyvien värivalintojen tekemistä koskevien ongelmien ratkaisemiseksi [Aupetit et al., 2014].

Parviälykkyystekniikoita on vuosien kuluessa sovellettu mitä moninaisimpiin ongelmiin. Perinteisesti ensimmäisten joukossa ratkaistavat sovellutukset ovat tunnettuja, vaikeasti laskettavaksi tiedettyjä, luonteeltaan didaktisia ongelmia. Tällaisia ovat kauppamatkustajan ongelman lisäksi muun muassa erilaiset aikataulutusongelmat, pakkausongelmat (bin packing) ja yleisesti graafien käsittelyyn liittyvät ongelmat, kuten maksimiklikkiongelma ja graafien väritysongelmat. Muita yleisiä sovellutuksia ovat muiden algoritmisten menetelmien, kuten neuroverkkojen, parametrisoinnin optimointi sekä bioinformatiikan alan optimointiongelmat. Sovellusalueet ovat jossain määrin riippuvaisia siitä, onko käsiteltävä optimointitekniikka suunniteltu pääasiassa kombinatoristen optimointiongelmien ratkaisuun: muurahaisyhdyskuntaoptimointia on helpompi soveltaa kombinatorisiin ongelmiin, kun taas partikkeliparvioptimointi on luontevammin sovellettavissa numeeriseen optimointiin. [Blum and Li, 2008]

Varsinkin jatkuvien optimointiongelmien osalta runsaasti sovelluskohteita tarjoavat insinööritieteet ja teollisuus, joiden asettamiin haasteisiin vastaaminen toimii suurena motivaationa optimointitekniikoiden kehittämisessä. Rakenteiden, kuten jousien ja hitsattujen rakenteiden, rasituskuorman optimointi näiden suunnittelussa on esimerkiksi yleinen sovelluskohde. Muun muassa käkihakua on onnistuneesti sovellettu rakenteellisen suunnittelun ongelmiin [Gandomi et al., 2013]. Myös muussa teollisuuden suunnittelussa optimoinnilla on suuri merkitys, kuten optiikkasuunnittelussa, jossa on sovellettu harmaasusioptimoijaa optisten signaalipuskurien rakenteen optimointiin. [Mirjalili et al., 2014]

Yksi menestyneimpiä parviälykkyysoptimointialgoritmien sovellutuksia on tietoverkkojen reititykseen kehitetty muurahaisyhdyskuntaoptimointiin perustuva AntNet. Sen kehittivät Di Caro ja Dorigo [1998] ja sitä on sittemmin sovellettu onnistuneesti käytännössä. Siitä tutkimuksessa tehdyt suorituskykymittaukset osoittavat sen olevan tarkoitukseensa nähden suorituskykyinen, vakaa ja muutoksiin hyvin reagoiva [Dhillon and Van Mieghem, 2007].

Parviälyn kiinnostavimpia kehitysalueita on parvirobotiikka, jonka tutkimuksen kehittämisessä on merkittävänä vaikuttajana toiminut muurahaisyhdyskuntaoptimoinnin Marco Dorigo. Parvirobotiikassa sovelletaan optimointialgoritmien mekanismeja siten, että parven yksilöt ovat fyysisiä robotteja, joiden keskinäisestä interaktiosta syntyy kollektiivista käyttäytymistä, joka johtaa keskittämättömään ongelmanratkaisuun robottiparvessa. Tutkimusala on alati kasvussa ja parvirobotiikka on osoittautunut virheensietokykyisesti ja monipuoliseksi ongelmanratkaisutekniikaksi fyysisessä maailmassa, joskin käytännön ongelmat, joita parvirobotiikan sovellutuksilla toistaiseksi kyetään ratkaisemaan, ovatkin vielä kovin yksinkertaisia. [Mondada et al., 2004]

Vaikka parviälykkyysalgoritmit ovat yleisesti hyvin lupaava tutkimuksen ala, liittyy parviälykkyyteen pohjaaviin optimointimenetelmiin tietynlaisia ongelmia. Ne eivät luonteestaan johtuen voi ensinnäkään olla koskaan absoluuttisen luotettavia. Heuristisiin menetelmiin perustuvina stokastisina approksimaatiomenetelminä niiden käyttäytymistä voi olla vaikea ennustaa odottamattomissa tilanteissa. Samaten koska ne reagoivat adaptiivisesti käsittelemäänsä ongelmaan, vaihtelee niiden käyttämä prosessointikuorma. Adaptiivisiin algoritmeihin suhtaudutaan muun muassa pankkisektorilla varovaisesti, jossa toimintavarmuutta pidetään kriittisenä. Toisekseen parviälykkyysalgoritmeilta puuttuu standardoidut suorituskykymittaristot, joiden avulla niiden suorituskykyä voitaisiin luotettavasti ja hyvällä vertailukelpoisuudella mitata. Nämä eivät ole kuitenkaan ylitsepääsemättömiä ongelmia. [Bonabeau et al., 1999]

Parviälykkyyteen pohjaavia optimointitekniikoita yhdistää moni asia. Ne ovat kaikki korkean tason heuristiikkoja ja osin määritelmäänkin perustuen stokastisia, populaatioperustaisia menetelmiä. Mekanismit, jotka ohjaavat optimointiprosessia, ovat kaikissa käsitellyissä algoritmeissa saaneet inspiraationsa luonnosta löytyvistä ja täten tosielämässä tehokkaiksi todetuista mekanismeista. Heuristiikat näyttäisivät olevan jaettavissa karkeasti kahteen ryhmään pohjatoiminnallisuutensa perusteella: yhtäällä ovat muurahaisyhdyskuntaoptimoinnin kaltaiset rakennusheuristiikat, joissa toiminta perustuu parveiluagenttien kulkemiseen hakuavaruutta mallintavan graafin poluilla jonkin heuristisen mallin pohjalta. Toista ryhmää edustavat partikkeliparvioptimoinnin kaltaiset numeerisen optimoinnin menetelmät, joissa toiminta perustuu hakuavaruudessa parveileviin vektoreihin, joiden suuntaa ja sijaintia päivitetään iteratiivisesti jonkin heuristisen mallin pohjalta. Vaikuttaisikin siltä, että näiden ryhmien sisällä optimointialgoritmit eroavat toisistaan eniten käyttöön valitun luonnonilmiön osalta ja siinä, millä tavoin tämä ilmiö on mallinnettu optimointitarkoitukseen.

Luonnosta löytyvien, evoluution luonnonvalinnan kautta hiottujen sovelluskelpoisten toimintamekanismien määrä on lukematon ja koska näitä mekanismeja optimointimenetelmiksi

soveltava tutkimusala on vireämpi kuin koskaan, tullaan uusia, monipuolisempia ja tehokkaampia parviälykkyysalgoritmeja näkemään varmasti mittavasti myös tulevaisuudessa.

Viiteluettelo

[Abdelkafi et al., 2014] Omar Abdelkafi, Julien Lepagnot and Lhassane Idoumghar, Multi-level parallelization for hybrid ACO. In: Patrick Siarry, Lhassane Idoumghar and Julien Lepagnot (eds.), *Swarm Intelligence Based Optimization*. Springer International Publishing, 2014, 60-67.

[Aupetit et al., 2014] Sébastian Aupetit, Nicolas Monmarché and Mohamed Slimane, Comparison of two swarm intelligence optimization algorithms on the textual color problem for web accessibility. In: Patrick Siarry, Lhassane Idoumghar and Julien Lepagnot (eds.), *Swarm Intelligence Based Optimization*. Springer International Publishing, 2014, 89-97.

[Beekman et al., 2008] Madeleine Beekman, Gregory A. Sword, Stephen J. Simpson, Biological foudations of swarm intelligence. In. Christian Blum and Daniel Merkle (eds.), *Swarm Intelligence, Introduction and Applications.* Springer Berlin Heidelberg, 2008, 3-41.

[Beni, 1988] Gerardo Beni, The concept of cellular robotic systems. In: *Proceedings of IEEE International Symposium on Intelligent Control*, 57-62.

[Bergh and Engelbrecht, 2006] F. van den Bergh and A.P. Engelbrech, A study of particle swarm optimization particle trajectories. *Information Sciences* **176**, 8 (Apr. 2006), 937-971.

[Blum, 2005] Christian Blum, Ant colony optimization: introduction and recent trends. *Physics of Life Reviews* **2** (Oct. 2005), 353–373.

[Blum and Li, 2008] Christian Blum and Xiadong Li, Swarm intelligence in optimization. In: Christian Blum and Daniel Merkle (eds.), *Swarm Intelligence, Introduction and Applications*. Springer Berlin Heidelberg, 2008, 43-85.

[Blum and Roli, 2003] Christian Blum and Andrea Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* **35**, 3 (Sept. 2003), 268-308.

[Bonabeau et al., 1999] Eric Bonabeau, Guy Theraulaz and Marco Dorigo, *Swarm Intelligence : From Natural to Artificial Systems*. Oxford University Press, 1999.

[Brown et al., 2007] Clifford T. Brown, Larry S. Liebovitch and Rahcel Glendon, Lévy flights in Dobe Ju/'hoansi foraging patterns. *Human Ecology* **35**, 1 (2007), 129-138.

[Bullnheimer et al., 1997] Bernd Bullnheimer, Richard F. Hartl and Christine Strauss, A new rank based version of the Ant System. A computational study. University of Vienna, Institute of Management Science. Technical report, 1997.

[Civicioglu and Besdok, 2011] Pinar Civicioglu and Erkan Besdok, A conceptual comparison of the Cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review* **39**, 4 (Apr. 2011), 315-346.

[Clerc, 1999] Maurice Clerc, The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99)* **3**, (1999).

[Clerc, 2010] Maurice Clerc, *Particle swarm optimization*. John Wiley & Sons, 2010.

[Cook, 1971] Stephen A. Cook, The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151-158.

[Deneubourg et al., 1990] Jean-Louis Deneubourg, Serge Aron, Simon Goss and Jacques Pasteels, The self-organizing exploratory pattern of the argentine ant. *Journal of Insect Behavior* **3,** 2 (Mar. 1990), 159–168.

[Dhillon and Van Mieghem, 2007] Santpal Singh Dhillon and Piet Van Mieghem, Peformance analysis of the AntNet algorithm. *Computer Networks* **51**, 8 (2007), 2104-2125.

[Di Caro and Dorigo, 1998] Gianni Di Caro and Marco Dorigo, AntNet, distributed stigmergic control for communications networks. *Journal of Artificial Intelligence Research* **9** (1998) 317-365.

[Dorigo, 1992] Marco Dorigo, *Optimization, learning and natural algorithms.* PhD Thesis, Diparimento di Electronica, Politecnico di Milano, Italy, 1992.

[Dorigo and Gambardelle, 1997] Marco Dorigo and Luca Maria Gambardelle, Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* **1**, 1 (1997), 53–66.

[Dorigo and Stützle, 2004] Marco Dorigo and Thomas Stützle, *Ant Colony Optimization*. MIT Press, 2004.

[Dorigo et al., 1999] Marco Dorigo, Gianni Di Caro and Luca Maria Gambardelle, Ant algorithms for discrete optimization. *Artificial Life* **5,** 2 (1999), 137–172.

[Drias et al., 2005] Habiba Drias, Souhila Sadeg and Safa Yahi, Cooperative bees swarm for solving the maximum satisfiability problem. In: John Cabestani, Alberto Prieto and Francisco Sandoval (eds.), *Computational Intelligence and Bioinspired Systems.* Springer Berlin Heidelberg, 2005, 318-325.

[Eberhart and Shi, 2000] Russ C. Eberhart and Yuhui Shi, Comparing inertia weights and constriction factors in particle swarm optimization. In: *Proceedings of the 2000 Congress on Evolutionary Computation* **1**, (2000), 84-88.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman & Co., San Francisco, 1979.

[Gandomi et al., 2013] Amir Hossein Gandomi, Xin-She Yang and Amir Hossein Alavi, Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. *Engineering with Computers* **29**, 1 (2013), 17-35.

[Garnier et al., 2007] Simon Garnier, Jacques Gautrais and Guy Theraulaz, The biological principles of swarm intelligence. *Swarm Intelligence* **1**, 1 (Jul. 2007), 3-31.

[Glover, 1986] Fred Glover, Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* **13**, 5 (1986), 533-549.

[Goss et al., 1989] Simon Goss, Serge Aron, Jean-Louis Deneubourg and Jacques Pasteels, Self-organized shortcuts in the Argentine ant. *Naturwissenschaften* **76**, 12 (Dec. 1989), 579–581.

[Grassé, 1946] Pierre-Paul Grassé, *Les Insectes Dans Leur Univers.* Ed. du Palais de la decouverte, Paris, France, 1946.

[Karaboga, 2005] Dervis Karaboga, An idea based on honey bee swarm for numerical optimization. Erciyes University, engineering faculty, computer engineering, Technical report-tr06 **200**, 2005.

[Karaboga and Akay, 2009] Dervis Karaboga and Bahriye Akay, A survey: algorithms simulating bee swarm intelligence. *Artificial Intelligence Review* **31**, 1-4 (Jun. 2009), 61-85.

[Karaboga and Basturk, 2007] Dervis Karaboga and Bahriy Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization,* **39**, 3 (2007) 459-471.

[Kennedy and Eberhart, 1995] James Kennedy and Russell Eberhart, Particle swarm optimization. In: *Proceedings of IEEE International Conference on Neural Networks*, **4** (1995), IEEE Press, 1942-1948.

[Kennedy and Eberhart, 1997] James Kennedy and Russel l Eberhart, A discrete binary version of the particle swarm algorithm. In: *Systems, Man, and Cybernetics, 1997 IEEE International Conference on Computational Cybernetics and Simulation* **5** (1997), 4104 - 4108.

[Krishnanand and Ghose, 2006] Kaipa N. Krishnanand and Debasish Ghose, Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. *Multiagent and Grid Systems* **2**, 3 (2006), 209-222.

[Laporte, 1992] Gilbert Laporte, The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* **59**, 2 (1992), 231-247.

[Liang and Suganthan, 2005] J.J. Liang, P. N. Suganthan, Dynamic multi-swarm particle swarm optimizer. In: *Proceedings of the IEEE Swarm Intelligence Symposium (SIS 2005),* 124-129.

[Mendes et al., 2004] Rui Mendes, James Kennedy and José Neves, The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation* **8**, 3 (2004), 204-210.

[Millonas, 1993] Mark M. Millonas, Swarms, phase transitions, and collective intelligence. In: Christopher G. Langton (ed.), *Artificial Life III.* Addison-Wesley, 1993, 417-445.

[Mirjalili et al., 2014] Seyedali Mirjalili, Seyed Mohammad Mirjalili and Andrew Lewis, Grey wolf optimizer. *Advances in Engineering Software*, **69** (Mar. 2014), 46-61.

[Mondada et al., 2004] Francesco Mondada, Giovanni C. Pettinaro, Andre Guignard, Iwo V. Kwee, Darino Floreano, Jean-Louis Deneubourg, Stefano Nolfi, Luca Maria Gambardelle and Marco Dorigo, SWARM-BOT: A new distributed robotic concept. *Autonomous Robots* **17**, 2-3 (2004), 193-221.

[Muro et al., 2011] C. Muro, R. Escobedo, L. Spector and R. Coppinger, Wolf-pack (Canis lupus) hunting strategies emerge from simple rules in computational simulations. *Behavioural Processes* **88**, 3 (Nov. 2011), 192-197.

[Niu and Wang, 2012] Ben Niu and Hong Wang, Bacterial colony optimization, *Discrete Dynamics in Nature and Society*, **2012**, 2012.

[Omran et al., 2014] Mahamad Omran, Maurice Clerc, Ayed Salman and Salah Alsharhan, A fuzzy-controlled comprehensive learning particle swarm optimizer. In: Patrick Siarry, Lhassane Idoumghar and Julien Lepagnot (eds.), *Swarm Intelligence Based Optimization*. Springer International Publishing, 2014, 35-41.

[Pham et al., 2006] Duc Pham, Afshin Ghanbarzadeh, Ebukekir Koç, Sameh Otri, S. Rahim, Muhamad Zaidi, The bees algorithm - a novel tool for complex optimisation problems. In: *Proceedings of the 2nd Virtual International Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, 454-459.

[Poli et al., 2007] Riccardo Poli, James Kennedy and Tim Blackwell, Particle swarm optimization. *Swarm Intelligence* **1**, 1 (Jul. 2007), 33-57.

[Rabanal et al., 2007] Pablo Rabanal, Ismael Rodríguez, Fernando Rubio, Using river formation dynamics to design heuristic algorithms. In: Selim G. Akl, Cristian S. Calude, Michael J. Dinneen, Grzegorz Rozenberg, H. Todd Wareham (eds.), *Unconventional Computation*. Springer Berlin Heidelberg, 2007, 163-177.

[Reynolds, 1987] Craig W. Reynolds, Flocks, herds and schools: A distributed behavioral model. *ACM Siggraph Computer Graphics* **21**, 4 (1987), 25-34.

[Storn and Price, 1997] Rainer Storn and Kenneth Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* **11** (1997), 341–359.

[Stützle and Hoos, 2000] Thomas Stützle and Holger H. Hoos, MAX-MIN ant system, *Future Generation Computer Systems* **16**, 8 (2000), 889-914.

[Teodorović and Dell'Orco, 2005] Dušan Teodorović and Mauro Dell'Orco, Bee colony optimization – a cooperative learing approach to complex problems. In: *Proceedings of the 16th Mini-EURO Conference and 10th meeting of EWGT* (2005), 51-60.

[Yang, 2005] Xin-She Yang, Engineering optimizations via nature-inspired virtual bee algorithms. In: José Mira and José Álvarez (eds.), *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach.* Springer Berlin Heidelberg, 2005, 317-323.

[Yang, 2010] Xin-She Yang, A new metaheuristic: bat-inspired algorithm. In: Juan R. González, David Alejandro Pelta, Carlos Cruz, Germán Terrazas and Natalio Krasnogor (eds.), *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*. Springer Berlin Heidelberg, 2010, 65-74.

[Yang, 2011] Xin-She Yang, Bat algorithm for multi-objective optimization, *International Journal of Bio-Inspired Computation* **3**, 5 (2011), 267-274.

[Yang and Deb, 2009] Xin-She Yang, Cuckoo search via Lévy flights. In: *World Congress on Nature & Biologically Inspired Computing (NaBIC 2009)*, 210-214.

[Zhu et al., 2010] Yan-fei Zhu, Xiong-Min Tang, Overview of swarm intelligence. In: *International Conference on Computer Application and System Modeling (ICCASM 2010)* **9** (2010), V9-400.